Build Database Clusters with MongoDB

Updated Monday, February 27, 2017 by Phil Zona

Written by Linode

Contribute on GitHub

MongoDB is a leading non-relational database management system, and a prominent member of the NoSQL (https://en.wikipedia.org/wiki/NoSQL) movement. Rather than using the tables and fixed schemas of a relational database management system (RDBMS), MongoDB uses key-value storage in collection of documents. It also supports a number of options for horizontal scaling in large, production environments. In this guide, we'll explain how to set up a *sharded cluster* for highly available distributed datasets.

There are two broad categories of scaling strategies for data. *Vertical scaling* involves adding more resources to a server so that it can handle larger datasets. The upside is that the process is usually as simple as migrating the database, but it often involves downtime and is difficult to automate. *Horizontal scaling* involves adding more servers to increase the resources, and is generally preferred in configurations that use fast-growing, dynamic datasets. Because it is based on the concept of adding more servers, not more resources on one server, datasets often need to be broken into parts and distributed across the servers. Sharding refers to the breaking up of data into subsets so that it can be stored on separate database servers (a sharded cluster).

The commands and filepaths in this guide are based on those used in Ubuntu 16.04 (Xenial). However, the configuration is the same for any system running MongoDB 3.2. To use this guide with a Linode running CentOS 7, for example, simply adjust the distro-specific commands and configuration files accordingly.

Before You Begin

1. To follow along with this guide, you will need at least six Linodes. Their functions will be explained in the next section. Follow our guides to install MongoDB (/docs/databases/mongodb/) on each Linode you want to use in your cluster.
2. Familiarize yourself with our Getting Started (/docs/getting-started/) guide and complete the steps for setting the hostname and timezone on each Linode. We recommend choosing hostnames that correspond with each Linode's role in the cluster, explained in the next section.
3. Complete the sections of our Securing Your Server (/docs/security/securing-your-server/) to create a standard user account, harden SSH access and remove unnecessary network services for each Linode.
4. Update your system:

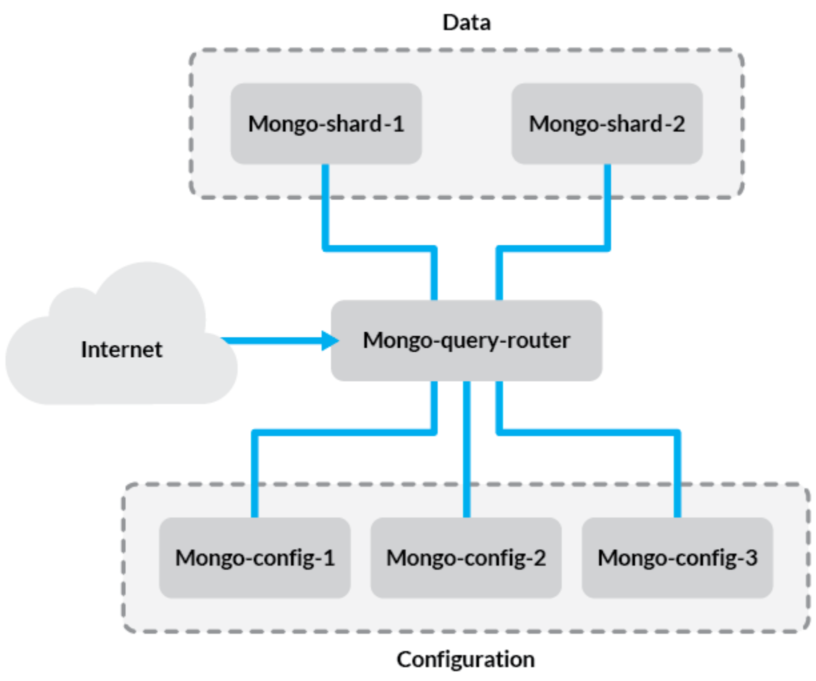
   ```
   sudo apt-get update && sudo apt-get upgrade
   ```

Note
This guide is written for a non-root user. Commands that require elevated privileges are prefixed with `sudo`. If you're not familiar with the `sudo` command, see the Users and Groups (/docs/tools-reference/linux-users-and-groups/) guide.

Cluster Architecture

Before we get started, let's review the components of the setup we'll be creating:

- **Config Server** - This stores metadata and configuration settings for the rest of the cluster. In this guide, we'll use one config server for simplicity but in production environments, this should be a replica set of at least three Linodes.
- **Query Router** - The `mongos` daemon acts as an interface between the client application and the cluster shards. Since data is distributed among multiple servers, queries need to be routed to the shard where a given piece of information is stored. The query router is run on the application server. In this guide, we'll only be using one query router, although you should put one on each application server in your cluster.
- **Shard** - A shard is simply a database server that holds a portion of your data. Items in the database are divided among shards either by range or hashing, which we'll explain later in this guide. For simplicity, we'll use two single-server shards in our example.



The problem in this configuration is that if one of the shard servers experiences downtime, a portion of your data will become unavailable. To avoid this, you can use replica sets (https://docs.mongodb.com/manual/reference/replica-configuration/) for each shard to ensure high availability. For more information, refer to our guide on creating MongoDB replica sets (/docs/databases/mongodb/create-a-mongodb-replica-set/).

Configure Hosts File

If your Linodes are all located in the same data center, we recommend adding a private IP address (/docs/networking/remote-access#adding-private-ip-addresses) for each one and using those here to avoid transmitting data over the public internet. If you don't use private IP addresses, be sure to encrypt your data with SSL/TLS (https://docs.mongodb.com/manual/tutorial/configure-ssl/).

On each Linode in your cluster, add the following to the `/etc/hosts` file:

/etc/hosts

```
192.0.2.1    mongo-config-1
192.0.2.2    mongo-config-2
192.0.2.3    mongo-config-3
192.0.2.4    mongo-query-router
192.0.2.5    mongo-shard-1
192.0.2.6    mongo-shard-2
```

Replace the IP addresses above with the IP addresses for each Linode. Also substitute the hostnames of the Linodes in your cluster for the hostnames above.

Note
You may also configure DNS records for each host rather than using hosts file entries. However, be aware that public DNS servers, such as the ones used when configuring records in the DNS Manager (/docs/platform/manager/dns-manager/), only support public IP addresses.

Set Up MongoDB Authentication

In this section you'll create a key file that will be used to secure authentication between the members of your replica set. While in this example you'll be using a key file generated with `openssl`, MongoDB recommends using an X.509 certificate (https://docs.mongodb.com/v3.2/core/security-x.509/) to secure connections between production systems.

Create an Administrative User

1. On the Linode that you intend to use as the *primary* member of your replica set of config servers, log in to the `mongo` shell:

   ```
   mongo
   ```

2. Connect to the `admin` database:

   ```
   use admin
   ```

3. Create an administrative user with `root` privileges. Replace "password" with a strong password of your choice:

   ```
   db.createUser({user: "mongo-admin", pwd: "password", roles:[{role: "root", db: "admin"}]})
   ```

Generate a Key file

1. Issue this command to generate your key file:

   ```
   openssl rand -base64 756 > mongo-keyfile
   ```

   Once you've generated the key, copy it to each member of your replica set.

2. The rest of the steps in this section should be performed on each member of the replica set, so that they all have the key file located in the same directory, with identical permissions. Create the `/opt/mongo` directory to store your key file:

   ```
   sudo mkdir /opt/mongo
   ```

3. Assuming that your key file is under the home directory for your user, move it to `/opt/mongo`, and assign it the correct permissions:

   ```
   sudo mv ~/mongo-keyfile /opt/mongo
   sudo chmod 400 /opt/mongo/mongo-keyfile
   ```

4. Update the ownership of your key file, so that it belongs to the MongoDB user. Use the appropriate command for your distribution:

   Ubuntu / Debian:

   ```
   sudo chown mongodb:mongodb /opt/mongo/mongo-keyfile
   ```

   CentOS:

   ```
   sudo chown mongod:mongod /opt/mongo/mongo-keyfile
   ```

5. Once you've added your key file, uncomment the `Security` section of the `/etc/mongod.conf` file on each of your Linodes, and add the following value:

   ```
   security:
   keyFile: /opt/mongo/mongodb-keyfile
   ```

   To apply the change, restart `mongod`:

   ```
   sudo systemctl restart mongod
   ```

   You can skip this step on your query router, since you'll create a separate configuration file for it later in this guide. Note that key file authentication automatically enables role-based access control (https://docs.mongodb.com/manual/core/authorization/), so you will need to create users (/docs/databases/mongodb/install-mongodb-on-ubuntu-16-04/#create-database-users) and assign them the necessary privileges to access databases.

Initialize Config Servers

In this section, we'll create a replica set of config servers. The config servers store metadata for the states and organization of your data. This includes information about the locations of data *chunks*, which is important since the data will be distributed across multiple shards.

Rather than using a single config server, we'll use a replica set to ensure the integrity of the metadata. This enables master-slave (primary-secondary) replication among the three servers and automates failover so that if your primary config server is down, a new one will be elected and requests will continue to be processed.

The steps below should be performed on each config server individually, unless otherwise specified.

1. On each config server, modify the following values in `/etc/mongod.conf` :

   /etc/mongod.conf

   ```
   port: 27019
   bindIp: 192.0.2.1
   ```

   The `bindIp` address should match the IP address you configured for each config server in your hosts file in the previous section. This should be a private IP address unless you've configured SSL/TLS encryption.

2. Uncomment the `replication` section and add the `replSetName` directive below it to create a replica set for your config servers:

   /etc/mongod.conf

   ```
   replication:
     replSetName: configReplSet
   ```

   `configReplSet` is the name of the replica set to be configured. This value can be modified, but we recommend using a descriptive name to help you keep track of your replica sets.

3. Uncomment the `sharding` section and configure the host's role in the cluster as a config server:

   /etc/mongod.conf

   ```
   sharding:
     clusterRole: "configsvr"
   ```

4. Restart the `mongod` service once these changes have been made:

   ```
   sudo systemctl restart mongod
   ```

5. On *one* of your config server Linodes, connect to the MongoDB shell over port 27019 with your administrative user:

   ```
   mongo mongo-config-1:27019 -u mongo-admin -p --authenticationDatabase admin
   ```

   Modify the hostname to match your own if you used a different naming convention than our example. We're connecting to the `mongo` shell on the first config server in this example, but you can connect to any of the config servers in your cluster since we'll be adding each host from the same connection.

6. From the `mongo` shell, initialize the replica set:

   ```
   rs.initiate( { _id: "configReplSet", configsvr: true, members: [ { _id: 0, host: "mongo-config-1:27019" }, { _id: 1, host: "mongo-conf
   ```

   Substitute your own hostnames if applicable. You should see a message indicating the operation succeeded:

   ```
   { "ok" : 1 }
   ```

7. Notice that the MongoDB shell prompt has now changed to `configReplSet:PRIMARY>` or `configReplSet:SECONDARY>` , depending on which Linode you used to run the previous commands. To further verify that each host has been added to the replica set:

   ```
   rs.status()
   ```

   If the replica set has been configured properly, you'll see output similar to the following:

```
configReplSet:SECONDARY> rs.status()
{
    "set" : "configReplSet",
    "date" : ISODate("2016-11-22T14:11:18.382Z"),
    "myState" : 1,
    "term" : NumberLong(1),
    "configsvr" : true,
    "heartbeatIntervalMillis" : NumberLong(2000),
    "members" : [
        {
            "_id" : 0,
            "name" : "mongo-config-1:27019",
            "health" : 1,
            "state" : 1,
            "stateStr" : "PRIMARY",
            "uptime" : 272,
            "optime" : {
                "ts" : Timestamp(1479823872, 1),
                "t" : NumberLong(1)
            },
            "optimeDate" : ISODate("2016-11-22T14:11:12Z"),
            "infoMessage" : "could not find member to sync from",
            "electionTime" : Timestamp(1479823871, 1),
            "electionDate" : ISODate("2016-11-22T14:11:11Z"),
            "configVersion" : 1,
            "self" : true
        },
        {
            "_id" : 1,
            "name" : "mongo-config-2:27019",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 17,
            "optime" : {
                "ts" : Timestamp(1479823872, 1),
                "t" : NumberLong(1)
            },
            "optimeDate" : ISODate("2016-11-22T14:11:12Z"),
            "lastHeartbeat" : ISODate("2016-11-22T14:11:17.758Z"),
            "lastHeartbeatRecv" : ISODate("2016-11-22T14:11:14.283Z"),
            "pingMs" : NumberLong(1),
            "syncingTo" : "mongo-config-1:27019",
            "configVersion" : 1
        },
        {
            "_id" : 2,
            "name" : "mongo-config-3:27019",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 17,
            "optime" : {
                "ts" : Timestamp(1479823872, 1),
                "t" : NumberLong(1)
            },
            "optimeDate" : ISODate("2016-11-22T14:11:12Z"),
            "lastHeartbeat" : ISODate("2016-11-22T14:11:17.755Z"),
            "lastHeartbeatRecv" : ISODate("2016-11-22T14:11:14.276Z"),
            "pingMs" : NumberLong(0),
            "syncingTo" : "mongo-config-1:27019",
            "configVersion" : 1
        }
    ],
    "ok" : 1
}
```

Configure Query Router

In this section, we'll set up the MongoDB query router. The query router obtains metadata from the config servers, caches it, and uses that metadata to send read and write queries to the correct shards.

All steps here should be performed from your query router Linode (this will be the same as your application server). Since we're only configuring one query router, we'll only need to do this once. However, it's also possible to use a replica set of query routers. If you're using more than one (i.e., in a high availability setup), perform these steps on each query router Linode.

1. Create a new configuration file called `/etc/mongos.conf`, and supply the following values:

   /etc/mongos.conf

   ```
   # where to write logging data.
   systemLog:
   destination: file
   logAppend: true
   path: /var/log/mongodb/mongos.log

   # network interfaces
   net:
   port: 27017
   bindIp: 192.0.2.4

   security:
   keyFile: /opt/mongo/mongodb-keyfile

   sharding:
   configDB: configReplSet/mongo-config-1:27019,mongo-config-2:27019,mongo-config-3:27019
   ```

   Replace `192.0.2.4` with your router Linode's private IP address, and save the file.

2. Create a new systemd unit file for `mongos` called `/lib/systemd/system/mongos.service`, with the following information:

   /lib/systemd/system/mongos.service

   ```
   [Unit]
   Description=Mongo Cluster Router
   After=network.target

   [Service]
   User=mongodb
   Group=mongodb
   ExecStart=/usr/bin/mongos --config /etc/mongos.conf
   # file size
   LimitFSIZE=infinity
   # cpu time
   LimitCPU=infinity
   # virtual memory size
   LimitAS=infinity
   # open files
   LimitNOFILE=64000
   # processes/threads
   LimitNPROC=64000
   # total threads (user+kernel)
   TasksMax=infinity
   TasksAccounting=false

   [Install]
   WantedBy=multi-user.target
   ```

   Note that the above example uses the `mongodb` user that MongoDB runs as by default on Ubuntu and Debian. If you're using CentOS, substitute the following values under the `Service` section of the file:

   ```
   [Service]
   User=mongod
   Group=mongod
   ```

3. The `mongos` service needs to obtain data locks that conflicts with `mongod`, so be sure `mongod` is stopped before proceeding:

```
sudo systemctl stop mongod
```

4. Enable `mongos.service` so that it automatically starts on reboot, and then initialize the `mongos`:

```
sudo systemctl enable mongos.service
sudo systemctl start mongos
```

5. Confirm that `mongos` is running:

```
systemctl status mongos
```

You should see output similar to this:

```
Loaded: loaded (/lib/systemd/system/mongos.service; enabled; vendor preset: enabled)
Active: active (running) since Tue 2017-01-31 19:43:05 UTC; 10s ago
Main PID: 3901 (mongos)
CGroup: /system.slice/mongos.service
    └─3901 /usr/bin/mongos --config /etc/mongos.conf
```

Add Shards to the Cluster

Now that the query router is able to communicate with the config servers, we must enable sharding so that the query router knows which servers will host the distributed data and where any given piece of data is located.

1. Log into *each* of your shard servers and change the following line in the MongoDB configuration file:

/etc/mongod.conf

```
bindIp: 192.0.2.5
```

The IP address in this line should be changed to the address corresponding with the one in your hosts file (since that's where address resolution will take place in our setup). For example, if you're using private IP addresses to connect your shards to the query router, use your private IP address. If you've configured SSL/TLS encryption and plan to use public IP addresses, use those.

2. From *one* of your shard servers, connect to the query router we configured above:

```
mongo mongo-query-router:27017 -u mongo-admin -p --authenticationDatabase admin
```

If your query router has a different hostname, substitute that in the command.

3. From the `mongos` interface, add each shard individually:

```
sh.addShard( "mongo-shard-1:27017" )
sh.addShard( "mongo-shard-2:27017" )
```

These steps can all be done from a single `mongos` connection; you don't need to log into each shard individually and make the connection to add a new shard. If you're using more than two shards, you can use this format to add more shards as well. Be sure to modify the hostnames in the above command if appropriate.

4. Optionally, if you configured replica sets (/docs/databases/mongodb/create-a-mongodb-replica-set/) for each shard instead of single servers, you can add them at this stage with a similar command:

```
sh.addShard( "rs0/mongo-repl-1:27017,mongo-repl-2:27017,mongo-repl-3:27017" )
```

In this format, `rs0` is the name of the replica set for the first shard, `mongo-repl-1` is the name of the first host in the shard (using port `27017`), and so on. You'll need to run the above command separately for each individual replica set.

> **Note**
> Before adding replica sets as shards, you must first configure the replica sets themselves.

Configure Sharding

At this stage, the components of your cluster are all connected and communicating with one another. The final step is to enable sharding. Enabling sharding takes place in stages due to the organization of data in MongoDB. To understand how data will be distributed, let's briefly review the main data structures:

- **Databases** - The broadest data structure in MongoDB, used to hold groups of related data.
- **Collections** - Analogous to tables in traditional relational database systems, collections are the data structures that comprise databases
- **Documents** - The most basic unit of data storage in MongoDB. Documents use JSON format for storing data using key-value pairs that can be queried by applications

Enable Sharding at Database Level

First, we'll enable sharding at the database level, which means that collections in a given database can be distributed among different shards.

1. Access the `mongos` shell on your query router. This can be done from any server in your cluster:

   ```
   mongo mongo-query-router:27017 -u mongo-admin -p --authenticationDatabase admin
   ```

   If applicable, substitute your own query router's hostname.
2. From the `mongos` shell, create a new database. We'll call ours `exampleDB`:

   ```
   use exampleDB
   ```
3. Enable sharding on the new database:

   ```
   sh.enableSharding("exampleDB")
   ```
4. To verify that the sharding was successful, first switch to the `config` database:

   ```
   use config
   ```

   Next, run a `find()` method on your databases:

   ```
   db.databases.find()
   ```

   This will return a list of all databases with some information about them. In our case, there should be only one entry for the `exampleDB` database we just created:

   ```
   { "_id" : "exampleDB", "primary" : "shard0001", "partitioned" : true }
   ```

Sharding Strategy

Before we enable sharding for a collection, we'll need to decide on a *sharding strategy*. When data is distributed among the shards, MongoDB needs a way to sort it and know which data is on which shard. To do this, it uses a *shard key*, a designated field in your documents that is used by the `mongos` query router know where a given piece of data is stored. The two most common sharding strategies are range-based and hash-based.

Range-based sharding divides your data based on specific ranges of values in the shard key. For example, you may have a collection of customers and associated address. If you use range-based sharding, the ZIP code may be a good choice for the shard key. This would distribute customers in a specified range of ZIP codes on the same shard. This may be a good strategy if your application will be running queries to find customers near each other when planning deliveries, for example. The downside to this is if your customers are not evenly distributed geographically, your data storage may rely too heavily on one shard, so it's important to analyze your data carefully before choosing a shard key. Another important factor to consider is what kinds of queries you'll be running, however. When properly utilized, range-base sharding is generally a better option when your application will be performing many complex read queries.

Hash-based sharding distributes data by using a hash function on your shard key for a more even distribution of data among the shards. Suppose again that you have a collection of customers and addresses. In a hash-based sharding setup, you may choose a customer ID number, for example, as the shard key. This number is transformed by a hash function, and the results of the hashing are what determines which shard the data is stored on. Hash-based sharding is a good strategy in situations where your application will mostly perform write operations, or if your application needs only to run simple read queries like looking up only a few specific customers at a time.

This is not intended to be a comprehensive guide to choosing a sharding strategy. Before making this decision for a production cluster, be sure to analyze your dataset, computing resources, and the queries your application will run. For more information, refer to MongoDB's documentation on sharding (https://docs.mongodb.com/v3.2/sharding/).

Enable Sharding at Collection Level

Now that the database is available for sharding and we've chosen a strategy, we need to enable sharding at the collections level. This allows the documents within a collection to be distributed among your shards. We'll use a hash-based sharding strategy for simplicity.

Note
It's not always necessary to shard every collection in a database. Depending on what data each collection contains, it may be more efficient to store certain collections in one location since database queries to a single shard are faster. Before sharding a collection, carefully analyze its anticipated contents and the ways it will be used by your application.

1. Connect to the `mongo` shell on your query router if you're not already there:

   ```
   mongo mongo-query-router:27017 -u mongo-admin -p --authenticationDatabase admin
   ```

2. Switch to the `exampleDB` database we created previously:

   ```
   use exampleDB
   ```

3. Create a new collection called `exampleCollection` and hash its `_id` key. The `_id` key is already created by default as a basic index for new documents:

   ```
   db.exampleCollection.ensureIndex( { _id : "hashed" } )
   ```

4. Finally, shard the collection:

   ```
   sh.shardCollection( "exampleDB.exampleCollection", { "_id" : "hashed" } )
   ```

   This enables sharding across any shards that you added to your cluster in the Add Shards to the Cluster section.

Test Your Cluster

This section is optional. To ensure your data is being distributed evenly in the example database and collection we configured above, you can follow these steps to generate some basic test data and see how it is divided among the shards.

1. Connect to the `mongo` shell on your query router if you're not already there:

   ```
   mongo mongo-query-router:27017 -u mongo-admin -p --authenticationDatabase admin
   ```

2. Switch to your `exampleDB` database:

   ```
   use exampleDB
   ```

3. Run the following code in the `mongo` shell to generate 500 simple documents and insert them into `exampleCollection`:

   ```
   for (var i = 1; i <= 500; i++) db.exampleCollection.insert( { x : i } )
   ```

4. Check your data distribution:

```
db.exampleCollection.getShardDistribution()
```

This will output information similar to the following:

```
Shard shard0000 at mongo-shard-1:27017
 data : 8KiB docs : 265 chunks : 2
 estimated data per chunk : 4KiB
 estimated docs per chunk : 132

Shard shard0001 at mongo-shard-2:27017
 data : 7KiB docs : 235 chunks : 2
 estimated data per chunk : 3KiB
 estimated docs per chunk : 117

Totals
 data : 16KiB docs : 500 chunks : 4
 Shard shard0000 contains 53% data, 53% docs in cluster, avg obj size on shard : 33B
 Shard shard0001 contains 47% data, 47% docs in cluster, avg obj size on shard : 33B
```

The sections beginning with `Shard` give information about each shard in your cluster. Since we only added two shards there are only two sections, but if you add more shards to the cluster, they'll show up here as well. The `Totals` section provides information about the collection as a whole, including its distribution among the shards. Notice that distribution is not perfectly equal. The hash function does not guarantee absolutely even distribution, but with a carefully chosen shard key it will usually be fairly close.

5. When you're finished, delete the test data:

```
db.dropDatabase()
```

Next Steps

Before using your cluster in a production environment, it's important to configure a firewall to limit ports 27017 and 27019 to only accept traffic between hosts within your cluster. Additional firewall configuration will likely be needed depending on the other services you're running. For more information, consult our firewall guides (/docs/security/firewalls/).

You may also want to create a master disk image consisting of a full MongoDB installation and whatever configuration settings your application requires. By doing so, you can use the Linode Manager to dynamically scale your cluster as your data storage needs grow. You may also do this from the Linode CLI (/docs/platform/api/linode-cli/) if you'd like to automate the process. For more information, see our guide on Linode images (/docs/platform/disk-images/linode-images/).

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- MongoDB Documentation for Replica Sets (https://docs.mongodb.com/manual/reference/replica-configuration/)
- MongoDB Documentation for Master-Slave Replication (https://docs.mongodb.com/manual/core/master-slave/)
- MongoDB Documentation for Sharding (https://docs.mongodb.com/manual/sharding/)
- MongoDB Documentation for Auto Sharding Configuration (https://docs.mongodb.com/manual/sharding/)
- Configure MongoDB for SSL/TLS (https://docs.mongodb.com/manual/tutorial/configure-ssl/)

Join our Community

Find answers, ask questions, and help others. (https://www.linode.com/community/questions/)

comments powered by Disqus (http://disqus.com)

Write for Linode.

We're always expanding our docs. If you like to help people, can write, and have expertise in a Linux or cloud infrastructure topic, learn how you can contribute (/docs/contribute) to our library.

Get started in the Linode Cloud today.

Create an Account (https://manager.linode.com/session/signup)

Intl.: +1 609-380-7100 (tel:+16093807100)

Email us (mailto:support@linode.com)

(https://facebook.com/linode) (https://twitter.com/linode) (https://plus.google.com/+linode/) (https://linkedin.com/com