**Running a MongoDB cluster**

One of the major benefits MongoDB offers is the ability to run a database cluster across multiple computers. In this portion of the lab, you'll create a cluster and watch data replicate. MongoDB supports two different kinds of distribution: replicas and shards. Replicas are used to have copies of the data on multiple servers, so that different users can access different replicas. Shards have different portions of the database itself on different computers, again to achieve performance gains. Replicas are easier to set up than shards, so we'll set up replicas in this lab.

In practice, replicas are typically run on separate computers that communicate with each other. For purposes of this lab, you'll run all three replicas on your lab computer, each running in a different directory. That's because a firewall in place on campus prevents us from having the replicas actually work across lab computers. The good news is that the experience you'll have in setting up the replicas is *exactly the same* as it would be across multiple computers. The only difference is that all of the replicas are on the same disk drive, which means that there isn't an actual performance gain, but everything else will look the same.

1. Shut down the MongoDB server if it is running. Find the terminal window where you left it running and issue a ctrl-c on the keyboard. You can keep a backup of the ‘data/db’ folder for running the server in standalone mode again later or create new ‘tmp’ directory and copy the contents of db folder inside.  
   Note: All the following commands assume you have created a new ‘tmp’ folder and have created inside it three more folders (db1, db2, db3). Folder db1 can be an exact copy of your existing ‘data/db’ folder or be empty. Folders db2 and db3 must be empty. Also, it is assumed that you have downloaded the zip version of the MongoDB Community Server and you have started a terminal from inside the downloaded folder (e.g., C:\<path-to-folder>\mongodb-win32-x86\_64-windows-8.0.4> )
2. Start the server as a replica server. To do this, issue the following command in your server terminal window:

/usr/local/mongodb/bin/mongod --dbpath /tmp/db1 --replSet "rs0" --port 27017

Windows:

.\bin\mongod.exe --dbpath=".\tmp\db1" --replSet "rs0" --port 27017

This starts up the server again, but associated with a replica set we're calling rs0, and listens for connections on the default port of 27017.

1. In a second terminal window, navigate to the /tmp directory. Create a new database directory, and start up a second server on another port. For example:
2. cd /tmp
3. mkdir db2
4. /usr/local/mongodb/bin/mongod --dbpath /tmp/db2 --replSet "rs0" --port 27018  
     
   Windows:  
     
   .\bin\mongod.exe --dbpath=".\tmp\db2" --replSet "rs0" --port 27018

Shrink this terminal window and move it out of the way somewhere.

1. Do it all again a third time, in yet another window:
2. cd /tmp
3. mkdir db3
4. /usr/local/mongodb/bin/mongod --dbpath /tmp/db3 --replSet "rs0" --port 27019  
     
   Windows:  
     
   .\bin\mongod.exe --dbpath=".\tmp\db3" --replSet "rs0" --port 27019
5. In yet another terminal window, start up the MongoDB shell for the first server by running the following command:
6. /usr/local/mongodb/bin/mongosh --port 27017  
     
   Windows:  
     
   mongosh –-port 27017
7. **Once the MongoDB shell has started**, issue the following command to initialize the replica set, and to see its configuration state. With this and the commands that follow, watch the output that gets dumped out in the terminal windows you have left running with the servers! It's fun to see them respond.
8. rs.initiate()
9. rs.conf()
10. Then add the remaining replica members to the set. To do so, enter into the MongoDB shell the following two commands:
11. rs.add("localhost:27018")
12. rs.add("localhost:27019")

This should get the whole replica set up and going.

Use MongoDB Compass to connect to the cluster or Write JavaScript code (or other programming languages) to add a new database and record to the primary replica. Usually, the primary server is the one we started first (port 27017). However, it's possible that you might end up with a different one; you can check the primary server from mongosh with the following command:

rs.status()

MongoDB uses a procedure to choose a single primary from the group. If that's the case, you'll need to modify the line in your Python program that looks like this:

client = MongoClient()

to instead look like this:

client = MongoClient('localhost',27018) ### or, 27019 if that's the port where the primary is

Write code to query the record you have just inserted to see if it is there.

Now, we're going to kill the primary server, and show that the data did in fact replicate. Go to the terminal window for whichever server ended up being the primary. Again, that's likely the first one you started. In that server terminal window, issue a ctrl-c. Watch what happens in the server windows for the other servers. They'll continue to complain that the first server is down, but they'll also pick a new primary.

Write Python code to again query for the record you added, now going up against one of the other replicas. You should be able to find the record, even though the server you originally added it to is down.