Lab 6: Replication

In this lab we will create a replica set with 3 nodes and add data to it.

1. For this lab please create a separate working directory (e.g. /mongodb_replication) and copy the file `lab_06.js` into this directory.

2. Open a shell a create a different data folder for any of the three members of the replica set:

```
> mkdir 1
> mkdir 2
> mkdir 3
```

3. First start MongoDB as a standalone server using a shell pointing to the directory created in step 1. Please note that this time we will explicitly tell `mongod` which port to use as we will start two more instances later:

```
> mongod --dbpath 1 --port 27001 --smallfiles --oplogSize 50
```

4. Now open an additional shell and connect to MongoDB (preloading the user defined functions needed for this lab):

```
> mongo --port 27001 --shell lab_06.js
```

   Then call this function to populate the data base with some test data:

```
> init()
```

   To verify if the data base has been populated correctly, please verify as following:

```
> use lab_06
> db.foo.count()
5000
```

5. Now we convert this Standalone MongoDB instance into a replica set with only one node. To do so, please stop `mongod` and restart it, but this time with the additional option `--replSet`:

```
> mongod --dbpath 1 --port 27001 --replSet replSet1 --smallfiles
--oplogSize 50
```

6. Now we add two new nodes to our replica set. Thus, please open two new shells (originating from our working directory) and start one node in each shell as following:

```
> mongod --dbpath 2 --port 27002 --smallfiles --oplogSize 50 --
replSet replSet1

> mongod --dbpath 3 --port 27003 --smallfiles --oplogSize 50 --
replSet replSet1
```

Please note that these nodes use different ports but the same name of the replica set.

7. Now the three nodes of our replica sets are up and running and as final step we need to configure the replica set. In doing so, please connect via the Mongo Shell to the node running on port 27001 and run the following:

```
cfg = {
      "_id" : "replSet1",
      "version" : 1,
      "members" : [
            {
                  "_id" : 0,
                  "host" : "localhost:27001"
            },
            {
                  "_id" : 1,
                  "host" : "localhost:27002"
            },
            {
                  "_id" : 2,
                  "host" : "localhost:27003"
            }
      ]
}

rs.initiate(cfg);
```

8. Now all three nodes are part of the replica set and you will see that data starts to be synchronized and that the folders `2` and `3` fill be filled with data. Also you can have a look at the log information displayed in the shells of node 2 and 3. Once the synchronization has been finished, you should check the status of the replica set:

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

9. Of course, now we want to check if our data really is reliably stored in the replica set. Thus we connect via Mongo Shell to node 1 and shut it down:

```
> rs.stepDown(30)
```

Notice: You could also just stop the *mongod* process abruptly which would work equally, but when doing a planned administration tasks on a replica set, this is the preferred method.

10. Now the replica set will be automatically reconfigured and we can find out the new primary node by connecting to one of two remaining nodes and calling `rs.status()`. Now we reconfigure the replica set to constantly remove node 1 (running on port 27001):

```
> rs.remove("localhost:27001")
```

11. What we get is a replica set with two nodes, which is not recommended for a productive environment but is just fine for our test system. Finally we connect to the secondary node and have a look at the `oplog`:

```
> db.isMaster().ismaster

  false

> use local
> db.oplog.rs.stats()
> db.oplog.rs.find()
```