Implementing Replica Sets

This lesson will teach you how to implement replica sets in MongoDB and the various commands used in the process.

WE'LL COVER THE FOLLOWING

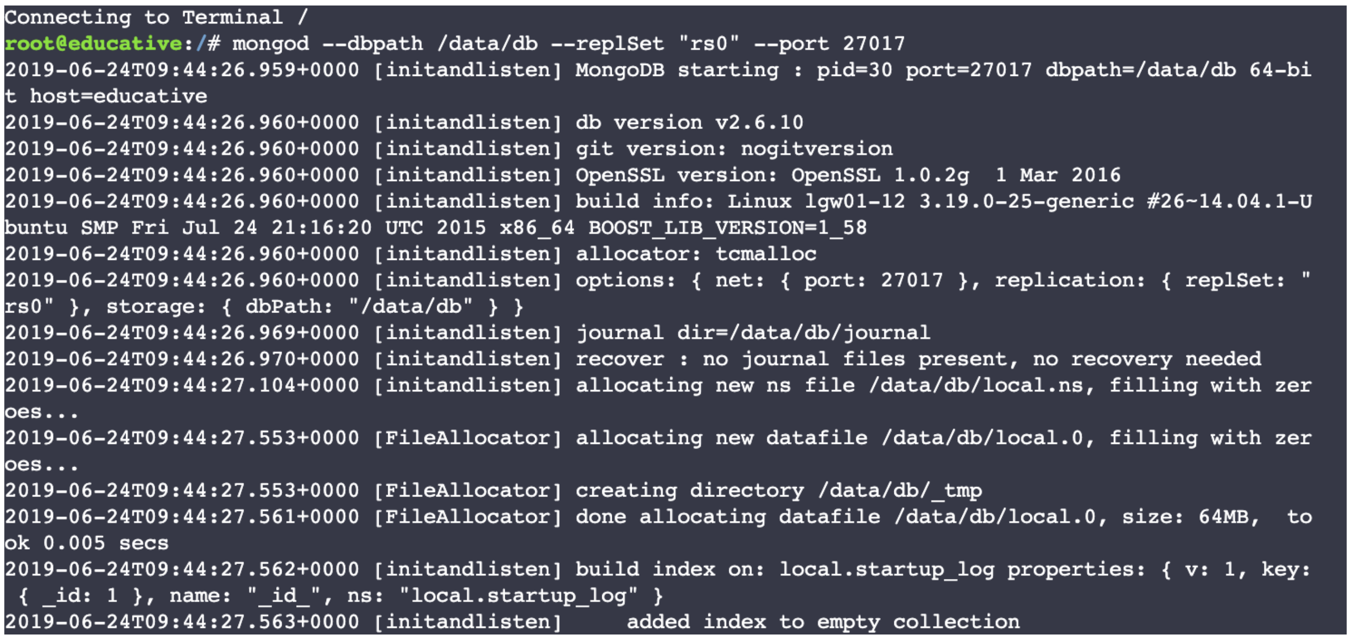
As in the previous chapter, the MongoDB shell client is used for demonstrative purposes.

> **Note:** Try running all of the commands mentioned below, one-by-one, in the terminal at the end of the lesson.

The three commands mentioned below, for running the mongod instances, have already been run for you in the terminal below.

Starting `mongod` Instances

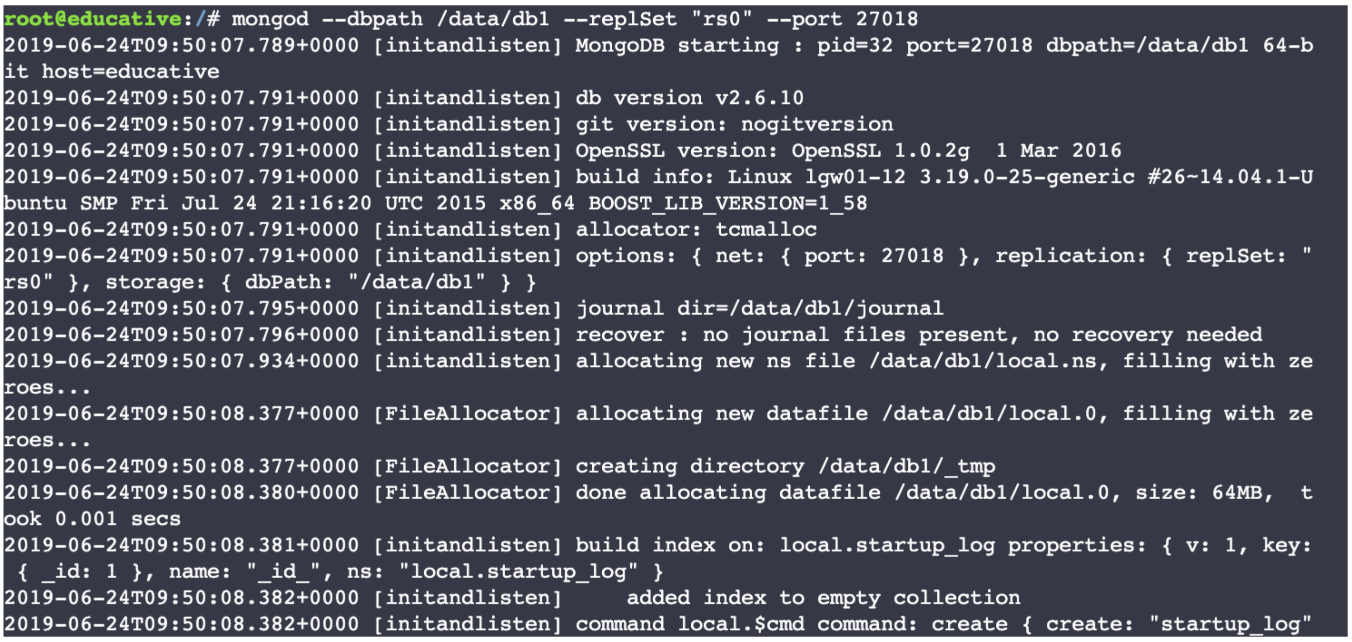
Here, we started one instance of `mongod` with some extra parameters:



```
Connecting to Terminal /
root@educative:/# mongod --dbpath /data/db --replSet "rs0" --port 27017
2019-06-24T09:44:26.959+0000 [initandlisten] MongoDB starting : pid=30 port=27017 dbpath=/data/db 64-bi
t host=educative
2019-06-24T09:44:26.960+0000 [initandlisten] db version v2.6.10
2019-06-24T09:44:26.960+0000 [initandlisten] git version: nogitversion
2019-06-24T09:44:26.960+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2019-06-24T09:44:26.960+0000 [initandlisten] build info: Linux lgw01-12 3.19.0-25-generic #26~14.04.1-U
buntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 BOOST_LIB_VERSION=1_58
2019-06-24T09:44:26.960+0000 [initandlisten] allocator: tcmalloc
2019-06-24T09:44:26.960+0000 [initandlisten] options: { net: { port: 27017 }, replication: { replSet: "
rs0" }, storage: { dbPath: "/data/db" } }
2019-06-24T09:44:26.969+0000 [initandlisten] journal dir=/data/db/journal
2019-06-24T09:44:26.970+0000 [initandlisten] recover : no journal files present, no recovery needed
2019-06-24T09:44:27.104+0000 [initandlisten] allocating new ns file /data/db/local.ns, filling with zer
oes...
2019-06-24T09:44:27.553+0000 [FileAllocator] allocating new datafile /data/db/local.0, filling with zer
oes...
2019-06-24T09:44:27.553+0000 [FileAllocator] creating directory /data/db/_tmp
2019-06-24T09:44:27.561+0000 [FileAllocator] done allocating datafile /data/db/local.0, size: 64MB,  to
ok 0.005 secs
2019-06-24T09:44:27.562+0000 [initandlisten] build index on: local.startup_log properties: { v: 1, key:
 { _id: 1 }, name: "_id_", ns: "local.startup_log" }
2019-06-24T09:44:27.563+0000 [initandlisten]     added index to empty collection
```

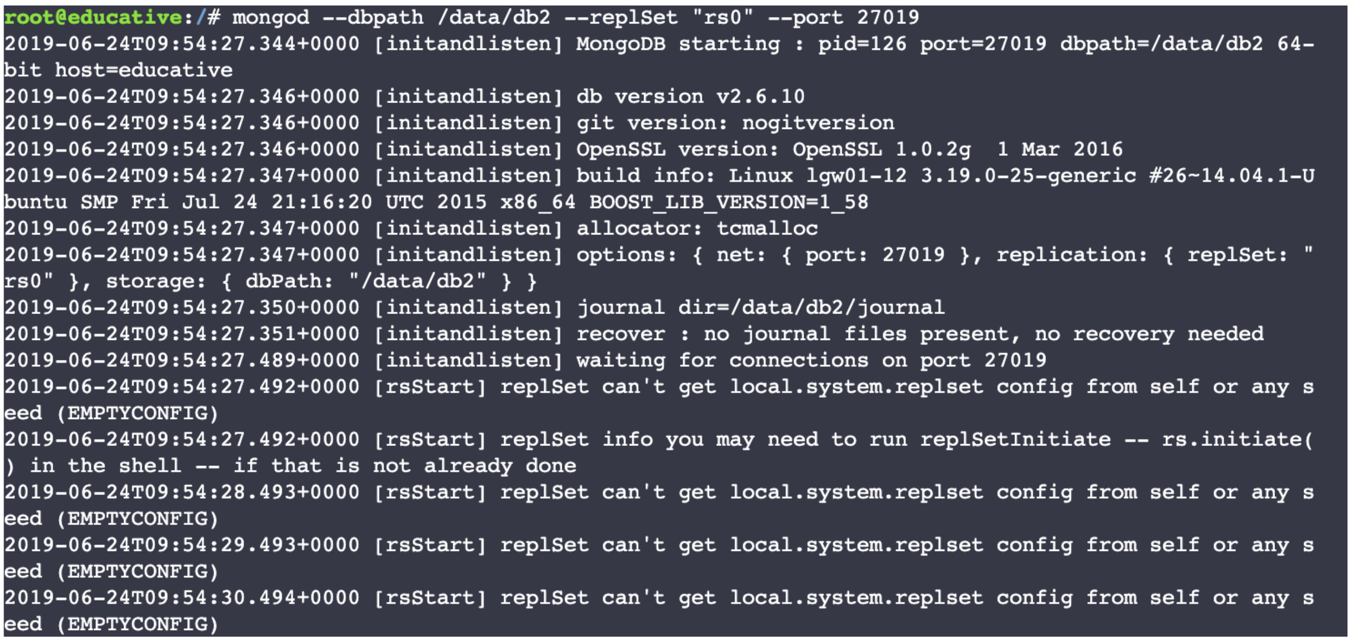
As you can see, we started it using extra parameter `--replSet`. Using this parameter, I defined a name for the replica set.

Now, I've done a similar call on port `27018`:



```
root@educative:/# mongod --dbpath /data/db1 --replSet "rs0" --port 27018
2019-06-24T09:50:07.789+0000 [initandlisten] MongoDB starting : pid=32 port=27018 dbpath=/data/db1 64-b
it host=educative
2019-06-24T09:50:07.791+0000 [initandlisten] db version v2.6.10
2019-06-24T09:50:07.791+0000 [initandlisten] git version: nogitversion
2019-06-24T09:50:07.791+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2019-06-24T09:50:07.791+0000 [initandlisten] build info: Linux lgw01-12 3.19.0-25-generic #26~14.04.1-U
buntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 BOOST_LIB_VERSION=1_58
2019-06-24T09:50:07.791+0000 [initandlisten] allocator: tcmalloc
2019-06-24T09:50:07.791+0000 [initandlisten] options: { net: { port: 27018 }, replication: { replSet: "
rs0" }, storage: { dbPath: "/data/db1" } }
2019-06-24T09:50:07.795+0000 [initandlisten] journal dir=/data/db1/journal
2019-06-24T09:50:07.796+0000 [initandlisten] recover : no journal files present, no recovery needed
2019-06-24T09:50:07.934+0000 [initandlisten] allocating new ns file /data/db1/local.ns, filling with ze
roes...
2019-06-24T09:50:08.377+0000 [FileAllocator] allocating new datafile /data/db1/local.0, filling with ze
roes...
2019-06-24T09:50:08.377+0000 [FileAllocator] creating directory /data/db1/_tmp
2019-06-24T09:50:08.380+0000 [FileAllocator] done allocating datafile /data/db1/local.0, size: 64MB,  t
ook 0.001 secs
2019-06-24T09:50:08.381+0000 [initandlisten] build index on: local.startup_log properties: { v: 1, key:
 { _id: 1 }, name: "_id_", ns: "local.startup_log" }
2019-06-24T09:50:08.382+0000 [initandlisten]     added index to empty collection
2019-06-24T09:50:08.382+0000 [initandlisten] command local.$cmd command: create { create: "startup_log"
```

And on port `27019`:



```
root@educative:/# mongod --dbpath /data/db2 --replSet "rs0" --port 27019
2019-06-24T09:54:27.344+0000 [initandlisten] MongoDB starting : pid=126 port=27019 dbpath=/data/db2 64-
bit host=educative
2019-06-24T09:54:27.346+0000 [initandlisten] db version v2.6.10
2019-06-24T09:54:27.346+0000 [initandlisten] git version: nogitversion
2019-06-24T09:54:27.346+0000 [initandlisten] OpenSSL version: OpenSSL 1.0.2g  1 Mar 2016
2019-06-24T09:54:27.347+0000 [initandlisten] build info: Linux lgw01-12 3.19.0-25-generic #26~14.04.1-U
buntu SMP Fri Jul 24 21:16:20 UTC 2015 x86_64 BOOST_LIB_VERSION=1_58
2019-06-24T09:54:27.347+0000 [initandlisten] allocator: tcmalloc
2019-06-24T09:54:27.347+0000 [initandlisten] options: { net: { port: 27019 }, replication: { replSet: "
rs0" }, storage: { dbPath: "/data/db2" } }
2019-06-24T09:54:27.350+0000 [initandlisten] journal dir=/data/db2/journal
2019-06-24T09:54:27.351+0000 [initandlisten] recover : no journal files present, no recovery needed
2019-06-24T09:54:27.489+0000 [initandlisten] waiting for connections on port 27019
2019-06-24T09:54:27.492+0000 [rsStart] replSet can't get local.system.replset config from self or any s
eed (EMPTYCONFIG)
2019-06-24T09:54:27.492+0000 [rsStart] replSet info you may need to run replSetInitiate -- rs.initiate(
) in the shell -- if that is not already done
2019-06-24T09:54:28.493+0000 [rsStart] replSet can't get local.system.replset config from self or any s
eed (EMPTYCONFIG)
2019-06-24T09:54:29.493+0000 [rsStart] replSet can't get local.system.replset config from self or any s
eed (EMPTYCONFIG)
2019-06-24T09:54:30.494+0000 [rsStart] replSet can't get local.system.replset config from self or any s
eed (EMPTYCONFIG)
```

Connecting Nodes

The next thing we want to do is put all these nodes together in one replica set.

In order to do this, we need to attach to one of the running `mongod` processes using the MongoDB shell client.

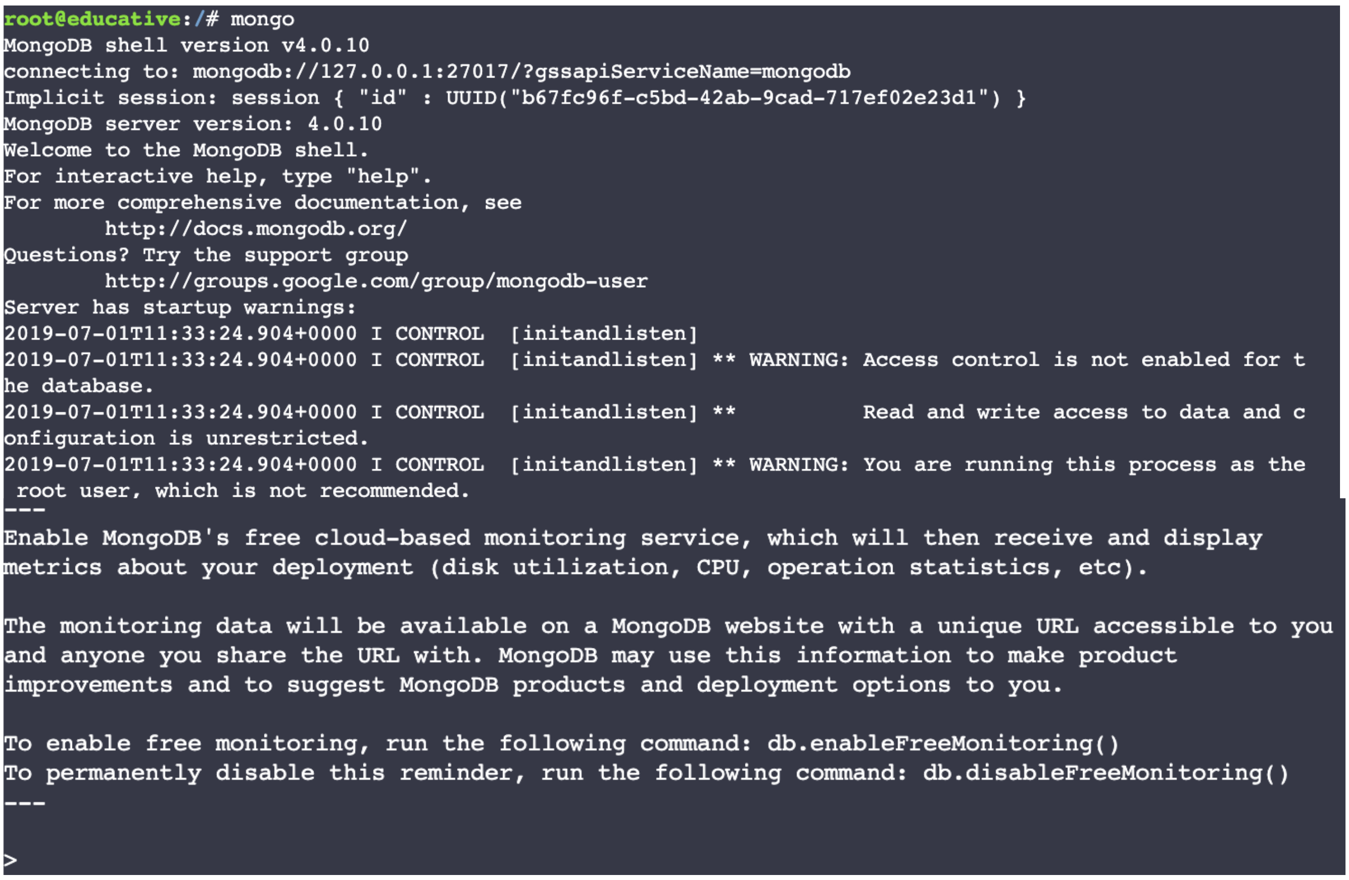
So, I called `mongo` to connect to the `mongod` process that is running on port `27017`.

Type the command, below, in the terminal:

```
mongo
```

Running "mongo" to connect to port 27017

You should see the following:



```
root@educative:/# mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("b67fc96f-c5bd-42ab-9cad-717ef02e23d1") }
MongoDB server version: 4.0.10
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2019-07-01T11:33:24.904+0000 I CONTROL  [initandlisten]
2019-07-01T11:33:24.904+0000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for t
he database.
2019-07-01T11:33:24.904+0000 I CONTROL  [initandlisten] **          Read and write access to data and c
onfiguration is unrestricted.
2019-07-01T11:33:24.904+0000 I CONTROL  [initandlisten] ** WARNING: You are running this process as the
 root user, which is not recommended.
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

Connecting mongo shell to port 27017

Using `rs.initiate()`

After that, I called the

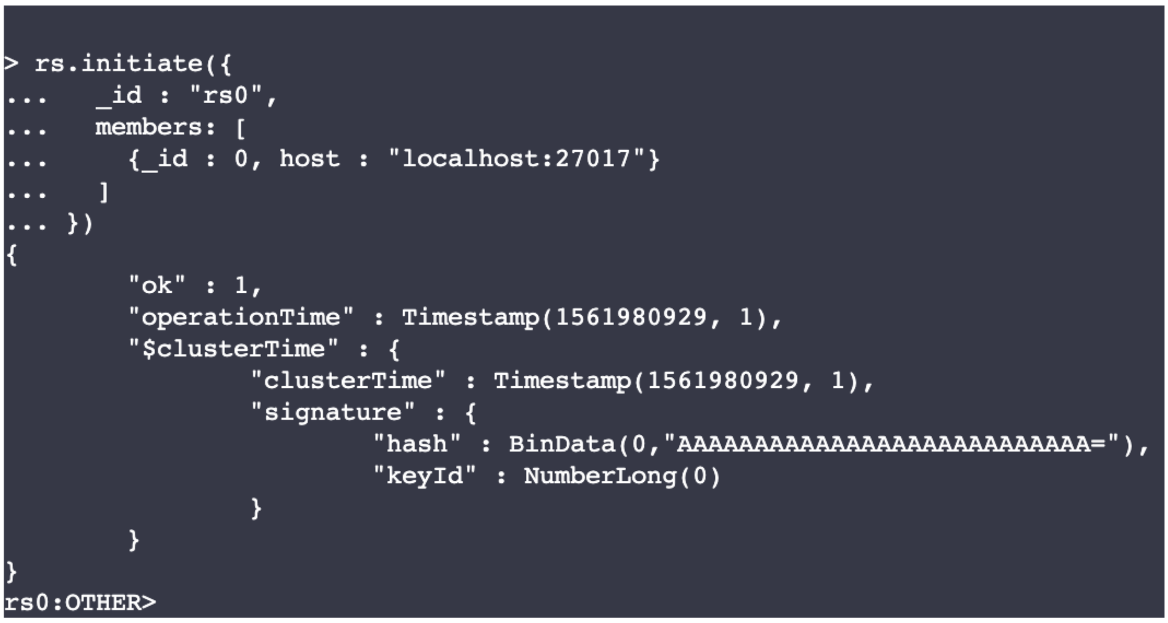
```
rs.initiate()
```

method:

```
rs.initiate({_id : "rs0",members: [{_id : 0, host : "localhost:27017"}]})
```

Calling rs.initiate() command

Run the above command in the terminal and you'll see the following result:



```
> rs.initiate({
...    _id : "rs0",
...    members: [
...      {_id : 0, host : "localhost:27017"}
...    ]
... })
{
        "ok" : 1,
        "operationTime" : Timestamp(1561980929, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1561980929, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:OTHER>
```

This function initiates the replica set and configures it with the passed JSON document (if one is passed into the function).

As you can see, I used this JSON document for configuration:

```
{
  _id : "rs0",
  members: [
    {_id : 0, host : "localhost:27017"}
  ]
}
```

In members, I just added just the node that is running on port `27017`. To add other nodes, you can simply extend this JSON to look like this:

```
{
    _id : "rs0",
    members: [
        { _id : 0, host : "localhost:27017" },
        { _id : 1, host : "localhost:27018" },
        { _id : 2, host : "localhost:27019" }
    ]
}
```

Using `rs.add()`

The other way this can be achieved is by calling the

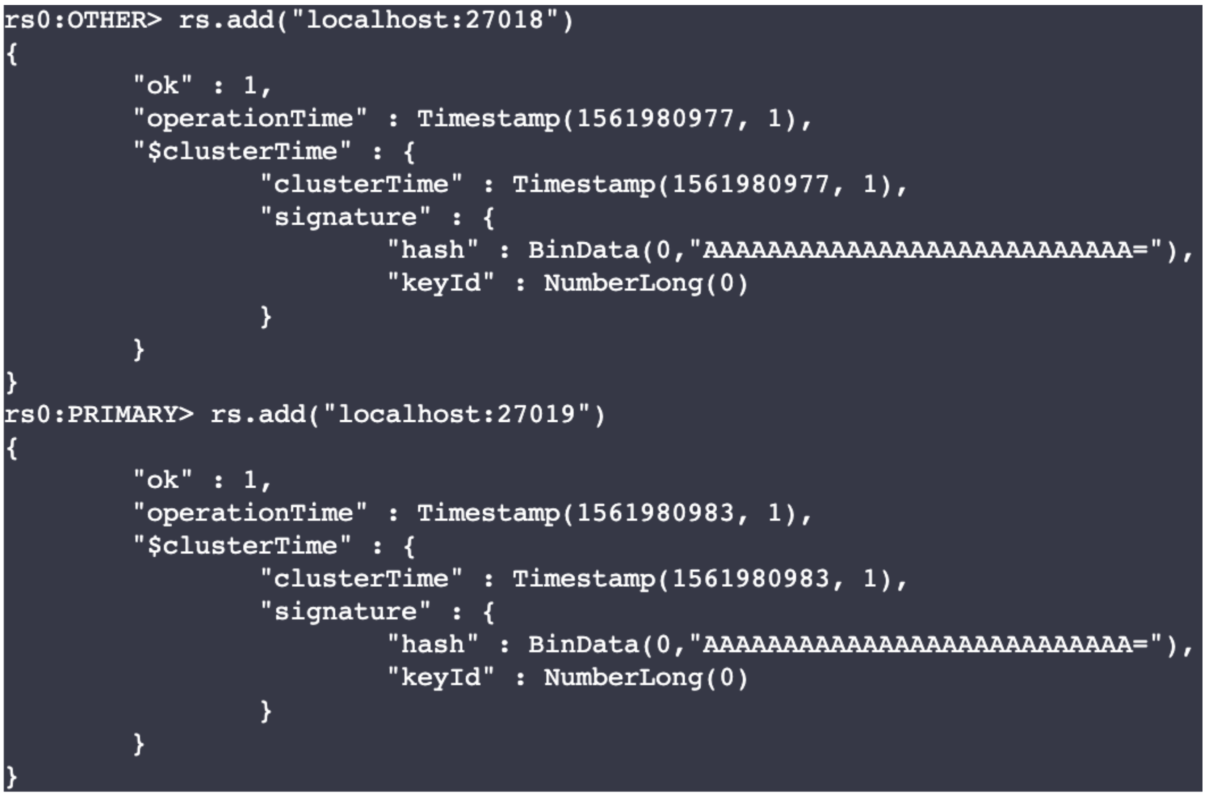
```
rs.add()
```

method, like so:

```
rs.add("localhost:27018")
rs.add("localhost:27019")
```

Running rs.add() commands for the other two ports

Run the above commands in the terminal and you'll see the following result:



```
rs0:OTHER> rs.add("localhost:27018")
{
        "ok" : 1,
        "operationTime" : Timestamp(1561980977, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1561980977, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:PRIMARY> rs.add("localhost:27019")
{
        "ok" : 1,
        "operationTime" : Timestamp(1561980983, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1561980983, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

Using `db.isMaster()`

Another useful function is:

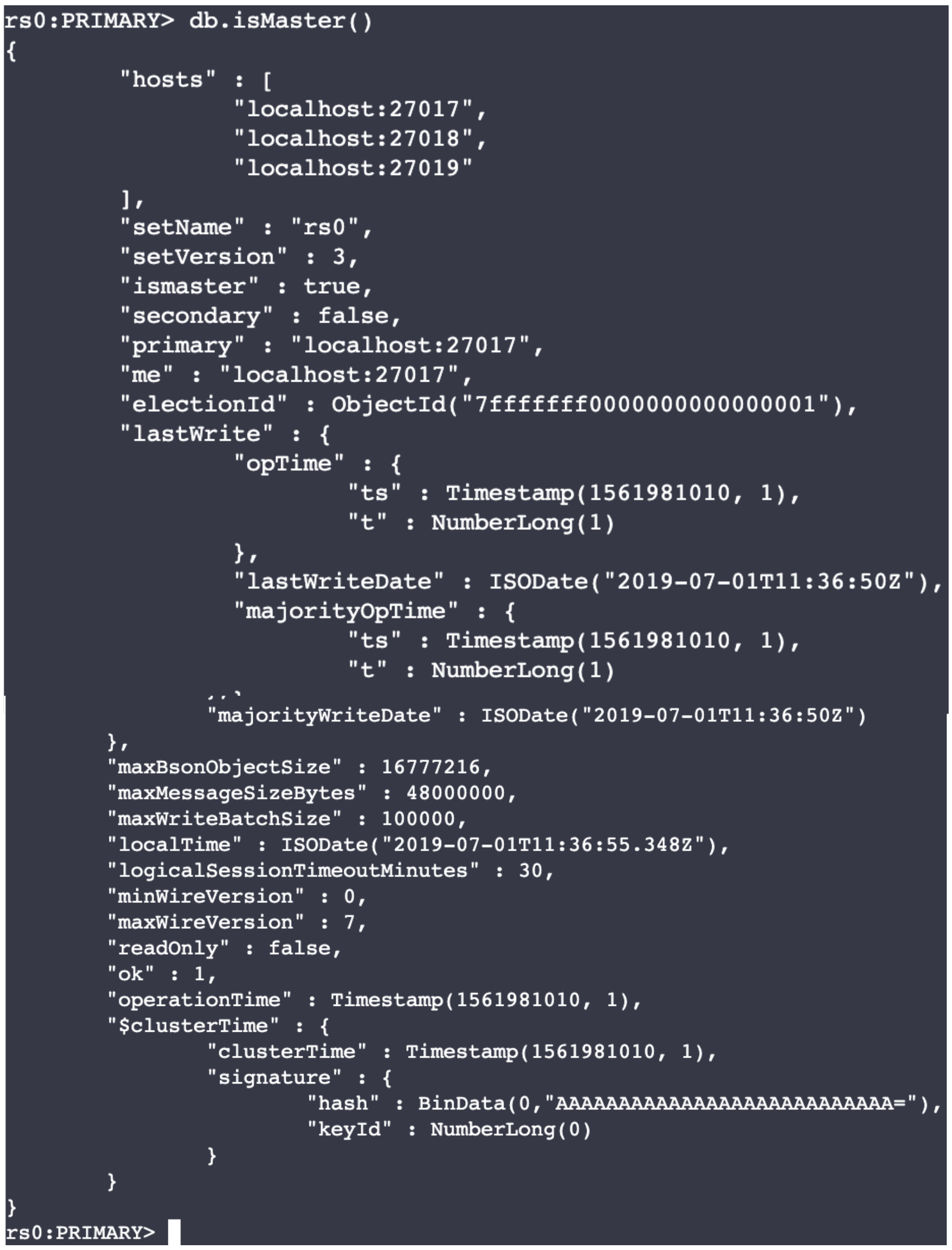
```
db.isMaster()
```

Try running this command in the terminal:

```
db.isMaster()
```

Running the db.isMaster() to display data about connected database

This function will show some interesting data about the database we are connected to, and about the replica set it belongs to:



```
rs0:PRIMARY> db.isMaster()
{
        "hosts" : [
                "localhost:27017",
                "localhost:27018",
                "localhost:27019"
        ],
        "setName" : "rs0",
        "setVersion" : 3,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "localhost:27017",
        "me" : "localhost:27017",
        "electionId" : ObjectId("7fffffff0000000000000001"),
        "lastWrite" : {
                "opTime" : {
                        "ts" : Timestamp(1561981010, 1),
                        "t" : NumberLong(1)
                },
                "lastWriteDate" : ISODate("2019-07-01T11:36:50Z"),
                "majorityOpTime" : {
                        "ts" : Timestamp(1561981010, 1),
                        "t" : NumberLong(1)
                },
                "majorityWriteDate" : ISODate("2019-07-01T11:36:50Z")
        },
        "maxBsonObjectSize" : 16777216,
        "maxMessageSizeBytes" : 48000000,
        "maxWriteBatchSize" : 100000,
        "localTime" : ISODate("2019-07-01T11:36:55.348Z"),
        "logicalSessionTimeoutMinutes" : 30,
        "minWireVersion" : 0,
        "maxWireVersion" : 7,
        "readOnly" : false,
        "ok" : 1,
        "operationTime" : Timestamp(1561981010, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1561981010, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
rs0:PRIMARY>
```

Here we can see:

- The name of the replica set
- Which node is the primary
- Which node we are connected to and so on

If you need more information about the replica set itself, use the

```
rs.status()
```

method, which will give you a more complex report.

Try out this command in the terminal:

Try out this command in the terminal.

```
rs.status()
```

Running rs.status() command

Terminal

> **Note:** Due to limitations of the platform, all three of the `mongod` commands will automatically be run for you in the terminal. You will, however, have to run the `mongo` command to start the shell, and then practice the rest of the commands discussed above.

● Terminal

Go back (click here to go back above).

In the next lesson, we will look at the durability of data in replica sets, in more detail.