

LAB: CREATING A MONGODB DATABASE

In this lab you will practice the basics of working with MongoDB to create a database and populate it with data. Tasks 1, 2 & 5 use a local installation of the MongoDB server, either on your computer or using the lab VM you can do all the tasks. Tasks 3 & 4 use MongoDB Atlas.

To use **mongo** (the client shell) and **mongoimport** you will need to have these tools installed on your computer (these are already installed in the lab VM). For MongoDB 4.4 or later you will need to download and install the MongoDB Shell and the MongoDB Database Tools from:

<https://www.mongodb.com/try/download/tools>

If you are using an older version, **mongoimport** should have been installed along with **mongod** and **mongo**, in the same location on your computer.

Task 1. Working with MongoDB

In this task you will learn to run a single MongoDB server instance and use the Mongo shell to create a database and insert documents in it.

1. Open a terminal window, and from your home directory start a MongoDB server instance with the command:

```
mongod --dbpath data/db
```

Note that **mongod** option names are preceded by a double dash (--). Note that you should enter commands manually rather than copy-paste from this PDF, as character encodings don't always transfer correctly when copying.

You should see a set of server startup messages which should include one indicating that the server is waiting for connections on port 27017.

Note that the above command assumes that the *bin* subdirectory of the MongoDB installation directory on your computer is on the current path, and that there is a valid data directory which is a subdirectory *data/db* in your home directory. Otherwise, you may have to include the full path to the *bin* directory, and you may have to create the data directory.

2. Open another terminal window and run the following command to start a Mongo shell client (again you may need to include the full path to the *bin* directory):

```
mongo
```

You should see a welcome message and a shell command prompt **>**, indicating that you have connected to the server.

3. Enter the following command in the shell, which will create a new database called *myblog* if it does not already exist, and switch to using that database:

```
> use myblog
```

4. Enter the following commands, which will store two new JSON documents in a **collection** *posts*. The collection will be created if it does not already exist. *db* is a reference to the current database.

```
> db.posts.insert({"title": "Using the MongoDB shell", "body": "Some info using the shell", "tags":["MongoDB", "shell", "getting started"]})
```

```
> db.posts.insert({"title": "Using PyMongo", "body": "Example of using the PyMongo Python driver", "tags":["MongoDB", "Python"]})
```

5. Enter the following command, which lists the collections in the current database – you should see *posts* listed.

```
> show collections
```

6. Enter the following command, which queries the database for all documents in the *posts* collection. The response you should see is shown in red here for clarity.

```
> db.posts.find()
{ "_id" : ObjectId("5897813afd25951dd45e830e"), "title" : "Using the MongoDB shell",
"body" : "Some info using the shell", "tags" : [ "MongoDB", "shell", "getting started" ] }
{ "_id" : ObjectId("58978154fd25951dd45e830f"), "title" : "Using PyMongo", "body" :
"Example of using the PyMongo Python driver", "tags" : [ "MongoDB", "Python" ] }
```

7. Enter the following commands which run simple queries on the *posts* collection:

```
> db.posts.find({"tags": "Python"})
```

```
> db.posts.find({"tags": "Python"}, {"title": 1})
```

What is the difference between the results of these two queries? You will practice more queries in another lab later in the module.

8. Run the following updates. Use **db.posts.find()** to check that the updates have been applied:

```
> db.posts.update({"title": "Using PyMongo"}, {$set: {"body": "New body"}})
```

```
> db.posts.update({"title": "Using PyMongo"}, {$push: {"tags": "New tag"}})
```

Describe the effect of each of these updates.

9. Enter the following command which creates a single key index, sorted in ascending order, on the *titles* field:

```
> db.posts.createIndex({"title": 1})
```

Based on the response, how many indexes did the collection have before running the command, and how many does it have now?

10. Enter the following command to list the indexes:

```
> db.posts.getIndexes()
```

What are the names of the indexes, and which fields do they index?

11. Enter the following command to delete a document. Check that the document has been deleted.

```
> db.posts.remove({"title": "Using PyMongo"})
```

12. Enter the following commands to delete the whole posts collection and then delete the myblog database:

```
> db.posts.drop()  
> db.dropDatabase()
```

13. Exit the shell and return to the system prompt using the command:

```
exit
```

Task 2: Importing data into MongoDB

In this task you will use the *mongoimport* utility to import data which has been saved as JSON text into MongoDB.

1. In your web browser, find the MongoDB “*restaurants*” sample database at the following URL, and save this document in your home directory as *primer-dataset.json*, or download the file from GCU Learn.

<https://raw.githubusercontent.com/mongodb/docs-assets/primer-dataset/primer-dataset.json>

2. Enter the following command at the system prompt to run the *mongoimport* utility and import this data into a collection *restaurants* in the database *test* (again you may need to include the full path to the *bin* directory):

```
mongoimport --db test --collection restaurants --drop --file primer-dataset.json
```

3. How many documents were imported? What do you think is the purpose of the *–drop* option?
4. Start the mongo shell again and make sure you are using the *test* database by entering:

```
use test
```

5. Enter a command to show the documents in the *restaurants* collection. Note that the shell shows these 20 at a time.
6. Enter a command to find the details of the restaurant with the name *Snack Time Grill*. What borough is it in?

Once you have completed the task drop the *restaurants* collection.

Note: in these labs you can do everything you need to do using the command line tools. However, there are a number of graphical tools available to work with MongoDB databases, for example Robo3T. This is installed on the Linux Mint VM, and you can start it by entering robo3t at a terminal prompt, and connecting to the database on localhost when the application starts up.

Task 3. Working in the cloud with MongoDB Atlas

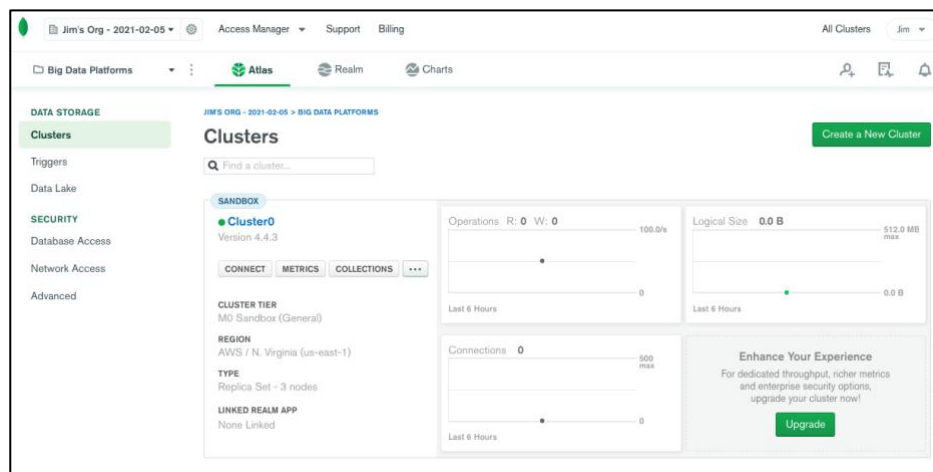
In this task you will learn to provision a MongoDB cluster in the cloud using the free tier of MongoDB Atlas, and connect to the cluster using the Mongo shell to create a database and insert documents in it.

If you don't have a MongoDB Atlas account, sign up at the following URL, using the Start free option:

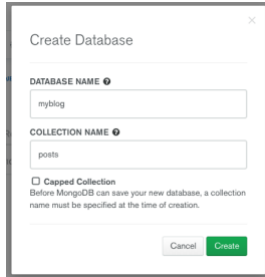
<https://www.mongodb.com/cloud/atlas>

No credit card is required, and you may be able to sign up with a Google account. Once signed in, choose an organisation and project name (e.g. "Big Data Platforms"). Choose a language if asked to do so, it doesn't matter what you choose but Python would be a good choice.


Choose to create a shared cluster, which is free. Be careful not to create a paid cluster! You can accept the defaults for the cluster options such as provider, region, MongoDB version, etc. When your cluster has been created you should see a view like the screenshot below (you should also see this view when you sign in in future). This view indicates the organisation and project name that you chose at sign-up. Now you are ready to start working with your cluster.




1. Click the **COLLECTIONS** button under the cluster name, which by default is Cluster 0. This will allow you to create a database and collection on your cluster. Click the **Add My Own Data** button. You should see a dialogue area overlaid on the web page. Choose *myblog* and *posts* as the names for the database and collection respectively and click **Create**.



Create Database

DATABASE NAME 

myblog

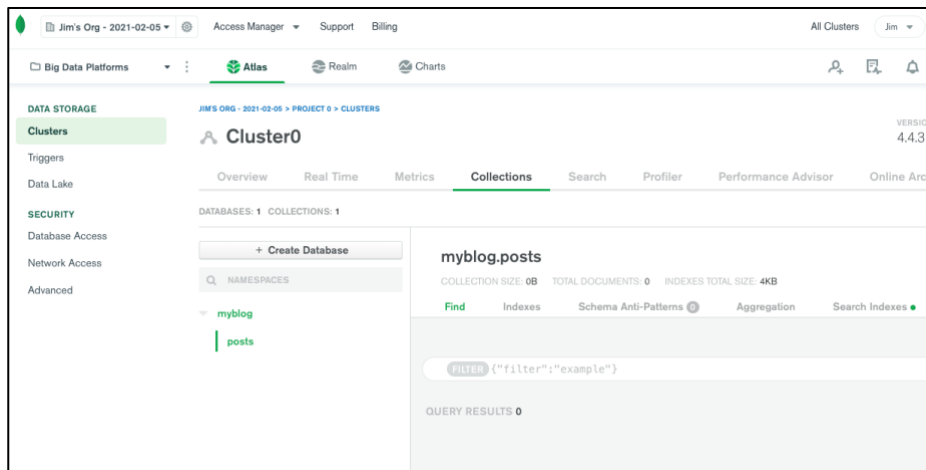
COLLECTION NAME 

posts

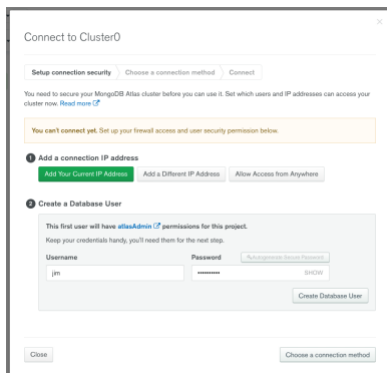
☐ Capped Collection
Before MongoDB can save your new database, a collection name must be specified at the time of creation.

Cancel Create

After a short delay you should see a view of your database, which will be currently empty.



- Click the **Clusters** menu item to return to the original view. Click the **CONNECT** button under the cluster name, which by default is *Cluster 0*. In the connect dialog firstly choose to add your current IP address to allow firewall access (you will have to do this again in future if you connect from other IP addresses – it is not recommended to *Allow access from anywhere*). You should also set a username and password for a database user – don't forget these, you will need them to connect.



Connect to Cluster0

Setup connection security Choose a connection method Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)


You can't connect yet. Set up your firewall access and user security permission below.


1 Add a connection IP address

Add Your Current IP Address Add a Different IP Address Allow Access from Anywhere

2 Create a Database User

This first user will have **atlasAdmin** permissions for this project. Keep your credentials handy, you'll need them for the next step.

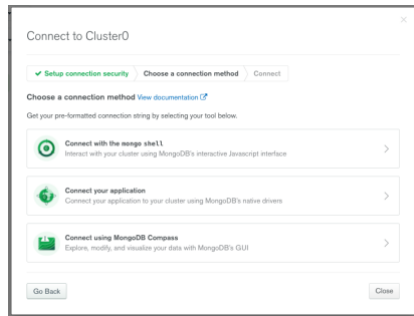
Username Password 

jm 

Create Database User

Close Choose a connection method

- Click the **Choose a connection method** to move to the next stage. Select **Connect with the mongo shell**. If you don't have the mongo shell installed you will see instructions on downloading and installing this for your OS.



NOTE: You can install the mongo shell without installing the server on your computer – if you do this it is recommended that you install the newer MongoDB shell (*mongosh*) from <https://www.mongodb.com/try/download/shell>.

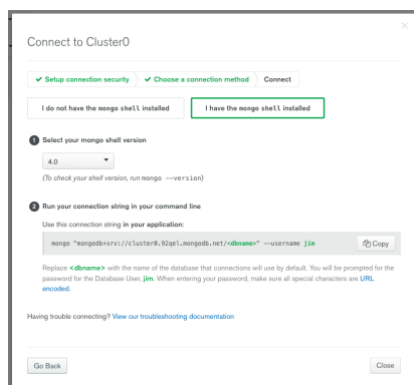
Don't use the *mongo* shell that is currently installed in the lab VM. If you want to connect from the lab VM, you can download and install MongoDB Shell on the VM.

- Once you have the shell installed, click the **I have the mongo shell installed** button. Select the shell version to match the version that you have installed. You will also see the connection string, which is the command you will run at your OS command prompt to run the shell and connect to the cluster. This will be something like one of the following, with the appropriate database name and username included.

mongo "mongodb+srv://cluster0.92qel.mongodb.net/<dbname>" --username <your username>

mongosh "mongodb+srv://cluster0.92qel.mongodb.net/<dbname>" --username <your username>

Copy the connection string.



5. Open a command prompt or terminal on your computer and paste the connection string as a command. Edit the command before executing to replace `<dbname>` with `myblog`. The username should already be the database user you set up. Execute the command. Note that this assumes you have the path on your OS set up to include the directory where you have the mongo executable installed. If not, you will have to edit the command to include the full path to mongo.
6. Enter the database user's password that you set up when prompted. The shell should start up and connect to the PRIMARY of the cluster. The shell prompt should look something like this:

MongoDB Enterprise atlas-5zcyn1-shard-0:PRIMARY>

Note that the MongoDB Atlas cluster is a replica set, and you are connecting to the primary node.

In the following steps you will interact with the cloud based cluster using the mongo shell. These commands repeat the steps in **Task 1** which demonstrated inserting and modifying data in a local MongoDB database.

7. Enter the following commands, which will store two new JSON documents in a **collection** `posts`. The collection will be created if it does not already exist. `db` is a reference to the current database.

```
> db.posts.insert({"title": "Using the MongoDB shell", "body": "Some info using the shell", "tags":["MongoDB", "shell", "getting started"]})
```

```
> db.posts.insert({"title": "Using PyMongo", "body": "Example of using the PyMongo Python driver", "tags":["MongoDB", "Python"]})
```

8. Enter the following command, which lists the collections in the current database – you should see `posts` listed.

```
> show collections
```

9. Enter the following command, which queries the database for all documents in the `posts` collection. The response you should see is shown in red here for clarity.

```
> db.posts.find()
{ "_id" : ObjectId("5897813afd25951dd45e830e"), "title" : "Using the MongoDB shell", "body" : "Some info using the shell", "tags" : [ "MongoDB", "shell", "getting started" ] }
{ "_id" : ObjectId("58978154fd25951dd45e830f"), "title" : "Using PyMongo", "body" : "Example of using the PyMongo Python driver", "tags" : [ "MongoDB", "Python" ] }
```

10. Enter the following commands which run simple queries on the posts collection:


```
> db.posts.find({"tags": "Python"})
```

```
> db.posts.find({"tags": "Python"}, {"title": 1})
```

What is the difference between the results of these two queries? You will practice more queries in another lab later in the module.

11. Run the following updates. Use **db.posts.find()** to check that the updates have been applied:

```
> db.posts.update({"title": "Using PyMongo"}, {$set: {"body": "New body"}})
```

```
> db.posts.update({"title": "Using PyMongo"}, {$push: {"tags": "New tag"}})
```

Describe the effect of each of these updates.

12. Enter the following command which creates a single key index, sorted in ascending order, on the titles field:

```
> db.posts.createIndex({"title": 1})
```

Based on the response, how many indexes did the collection have before running the command, and how many does it have now?

13. Enter the following command to list the indexes:

```
> db.posts.getIndexes()
```

What are the names of the indexes, and which fields do they index?

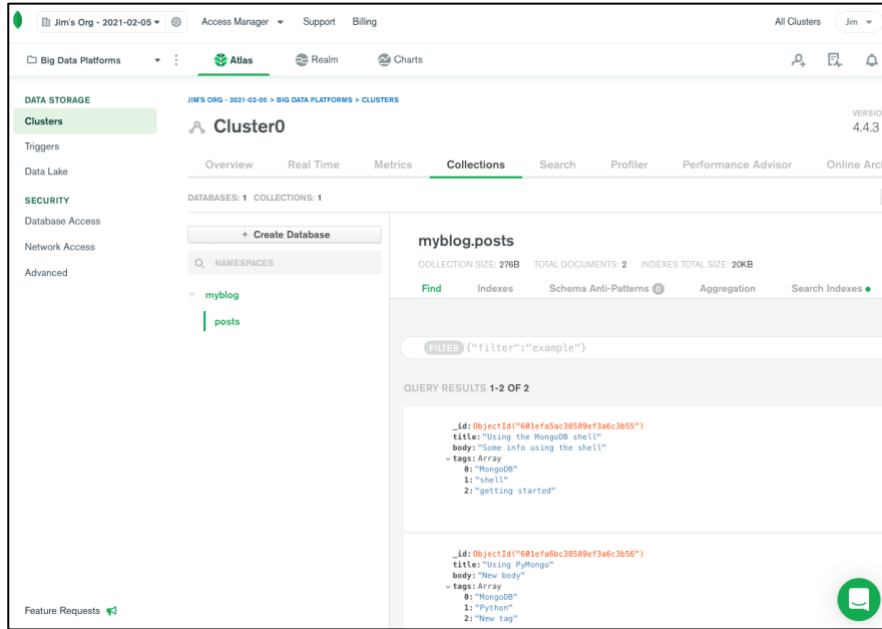
14. Enter the following command to delete a document. Check that the document has been deleted.

```
> db.posts.remove({"title": "Using PyMongo"})
```

15. Exit the shell and return to the system prompt using the command:

```
exit
```

16. In the MongoDB Atlas web interface click the **COLLECTIONS** button as before to see the contents of the database. You should see the documents that you have created. Note that you can insert and edit data and run queries within the web interface, but connecting using the shell gives you the ability to use all the query capabilities of MongoDB.



The screenshot displays the MongoDB Atlas web interface. The top navigation bar includes 'Jim's Org - 2021-02-05', 'Access Manager', 'Support', and 'Billing'. The main header shows 'Big Data Platforms', 'Atlas', 'Realm', and 'Charts'. The left sidebar contains 'DATA STORAGE' (Clusters, Triggers, Data Lake) and 'SECURITY' (Database Access, Network Access, Advanced). The main content area is titled 'Cluster0' and shows 'DATABASES: 1' and 'COLLECTIONS: 1'. A 'myblog' database is expanded, showing a 'posts' collection. The 'posts' collection details include 'COLLECTION SIZE: 276B', 'TOTAL DOCUMENTS: 2', and 'INDEXES TOTAL SIZE: 20KB'. A filter is applied: 'filter: "example"'. The query results show two documents:

```
1. {
  "_id": ObjectId("601efaf0c38599ef3a6c3b55"),
  "title": "Using the MongoDB shell",
  "body": "Some info using the shell",
  "tags": Array
    0: "MongoDB"
    1: "shell"
    2: "getting started"
}
```

```
2. {
  "_id": ObjectId("601efaf0c38599ef3a6c3b56"),
  "title": "Using PyMongo",
  "body": "New body",
  "tags": Array
    0: "MongoDB"
    1: "Python"
    2: "New tag"
}
```

Task 4: Importing data into MongoDB Atlas

For this task, although you will continue to work in the cloud with MongoDB, you will need the `mongoimport` tool installed locally (see the note on installing local tools at the start of this lab). In this task you will simply practice importing the contents of a JSON file into MongoDB Atlas.

In MongoDB Atlas UI:

1. Take note of the username and password of your database user. In the example commands here, assume these are *jim* and *Pa\$\$w0rd*, you should use your own values
2. In the **Collections** tab create a new collection in the database that you created in Lab 2. In this example, assume the database is called *bdp* and the collection is called *companies*.
3. Get the connection string as you did in Task 3. For this example assume this is "**mongodb+srv://cluster0.abc1a.mongodb.net/bdp**", you should use your own value.

In a command prompt or terminal on your computer:

1. Copy your data file to your current working directory. In this example, the file is called *companies.json*, which you can download from GCU Learn.
2. Execute the following command - note that this assumes you have the path on your OS set up to include the directory where you have the `mongoimport` executable installed. If not, you will have to edit the command to include the full path to `mongoimport`. Note that the username and password of your database user are embedded in the URI.s

```
mongoimport --uri="mongodb+srv://jim:Pa$$w0rd@cluster0.abc1a.mongodb.net/bdp" --collection=companies --file=companies.json
```

You **should** see output similar to the following:

```
2021-02-23T21:16:27.697+0000 connected to: atlas-abc1a-shard-0/cluster0-shard-00-00.abc1a.mongodb.net:27017,cluster0-shard-00-01.abc1a.mongodb.net:27017,cluster0-shard-00-02.abc1a.mongodb.net:27017
2021-02-23T21:16:30.191+0000 [###.....] bdp.companies 9.39MB/74.6MB (12.6%)
2021-02-23T21:16:33.191+0000 [####.....] bdp.companies 14.8MB/74.6MB (19.8%)
2021-02-23T21:16:36.191+0000 [#####.....] bdp.companies 19.3MB/74.6MB (25.8%)
2021-02-23T21:16:39.191+0000 [#####.....] bdp.companies 26.6MB/74.6MB (35.6%)
2021-02-23T21:16:42.192+0000 [#####.....] bdp.companies 30.0MB/74.6MB (40.3%)
2021-02-23T21:16:45.192+0000 [#####.....] bdp.companies 37.5MB/74.6MB (50.2%)
2021-02-23T21:16:48.192+0000 [#####.....] bdp.companies 43.9MB/74.6MB (58.9%)
2021-02-23T21:16:51.195+0000 [#####.....] bdp.companies 50.9MB/74.6MB
```

```

(68.2%)
2021-02-23T21:16:54.192+0000 [#####.....] bdp.companies 54.9MB/74.6MB
(73.5%)
2021-02-23T21:16:57.194+0000 [#####.....] bdp.companies 61.3MB/74.6MB
(82.2%)
2021-02-23T21:17:00.196+0000 [#####..] bdp.companies 68.8MB/74.6MB
(92.1%)
2021-02-23T21:17:03.192+0000 [#####] bdp.companies 74.6MB/74.6MB
(100.0%)
2021-02-23T21:17:04.293+0000 [#####] bdp.companies 74.6MB/74.6MB
(100.0%)
2021-02-23T21:17:04.293+0000 imported 18801 documents

```

You can now connect to your MongoDB Atlas cluster with `mongo` to check that the collection contains data, and explore that data with `db.find` commands (or you can connect with Robo3T). Alternatively, you can explore the data in the **Collections** tab in the MongoDB Atlas UI.

Log out of MongoDB Atlas and close your command prompt or terminal window.

Task 5: Setting up a MongoDB replica set

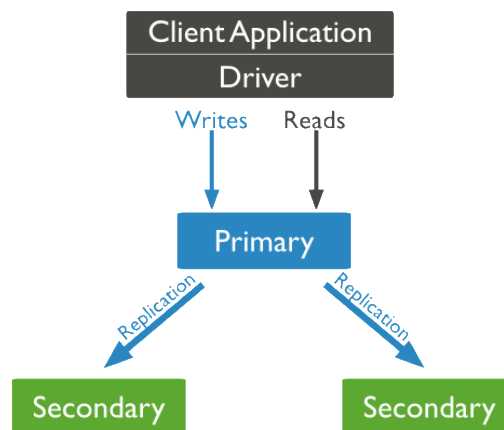
This task should be attempted on your own computer or the lab VM.

In the previous tasks you have worked with a single instance of MongoDB (and you will do the same in most of the lab exercises to come on MongoDB). However, MongoDB is designed primarily to be used as a distributed database, with multiple instances configured as a cluster, making use of:

- **replication** - for redundancy and high availability
- **sharding** - to distribute a large set of data across multiple nodes, each with its own subset of the data

In this task you will implement and observe replication between two instances. For simplicity, these instances will both run on a single machine, although in a real-world scenario these would be on separate machines to offer redundancy.

You will create a replica set with a Primary instance and a single Secondary instance, although in principle you can add as many Secondaries as you like.



1. Open a terminal window, and create a directory *data1* in your home directory, and inside that create a directory *db*. Create another directory *data2* in your home directory, also with a directory *db* inside it.
2. From your home directory start a MongoDB server instance with the command:

```
mongod --port 27017 --dbpath data1/db --replSet rs0
```

You should see a set of server startup messages which should include one indicating that the server is waiting for connections on port 27017.

3. Open another terminal window and start another MongoDB server instance, listening on a different port but able to be part of the same replica set, with the command:

```
mongod --port 27018 --dbpath data2/db --replSet rs0
```

Note that each instance has its own separate data directory.

4. Open another terminal window and run the following command to start a client:

mongo

This will connect to the default port 27017, so will access the first MongoDB instance.

5. At the client shell prompt, enter the following command to initiate replication – you should see the response shown:

```
rs.initiate()  
{  
  "info2" : "no configuration specified. Using a default  
configuration for the set",  
  "me" : "osboxes:27017",  
  "ok" : 1  
}
```

You should see the client shell prompt change to:

rs0:PRIMARY>

Note that it may appear as SECONDARY initially until your instance is “elected” as Primary.

6. Now add the secondary instance to the replica set using the following command. You need to specify the hostname and port of that instance. On the Linux Mint VM the host name is *osboxes*, so you may need to modify the command if you are using a different host machine.

```
rs0:PRIMARY> rs.add("osboxes:27018");  
{ "ok" : 1 }
```

You can check the status of the replica set as follows. You should see both instances listed as members in the output (not shown here as it is quite verbose, but you should be able to find the relevant information when you run the command).:

rs0:PRIMARY> rs.status()

7. Now you can test replication by inserting a data item in the Primary, and observing that the same data is replicated automatically to the Secondary.

Run the following commands at the client prompt to insert an object in a collection seasonal in a database fruits:

```
rs0:PRIMARY> use fruits  
switched to db fruits
```

```
rs0:PRIMARY> db.seasonal.insertOne({name:"Mango",season:"Summer"})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5c4dde81dbb1cb9072f347ba")
}
```

8. Now open another terminal window and start a client connected to the Secondary instance with the command:

```
mongo --port 27018
```

The shell prompt should appear as **rs0:SECONDARY>**

9. Enter the following commands to list the contents of the same collection as you inserted the data into on the Primary. You should see an object containing the data you inserted on the Primary.

```
rs0:SECONDARY> rs.slaveOk()
```

```
rs0:SECONDARY> use fruits
```

```
switched to db fruits
```

```
rs0:SECONDARY> show collections
```

```
seasonal
```

```
rs0:SECONDARY> db.seasonal.find();
```

```
{ "_id" : ObjectId("5c4dde81dbb1cb9072f347ba"), "name" : "Mango", "season" :
"Summer" }
```

This demonstrates that replication is taking place successfully within your simple replica set!

10. Other things to try while your replica set is running:

- Try to insert a document into the seasonal collection from the client connected to the secondary – is this allowed?
- Shut down the primary server (Ctrl-C in its terminal window) – do you see any change to the command prompt in the client connected to the secondary?

For more information on configuring replication and sharding on MongoDB clusters, see the MongoDB documentation:

<https://docs.mongodb.com/manual/replication/>

<https://docs.mongodb.com/manual/sharding/>