Install MongoDB 8.0.4 on Windows OS

Table of Contents

This document outlines the steps needed to install **MongoDB** and configure 2 types of clusters – **Replica Set** and **Sharded Cluster** – on Windows Operating system and work with MongoDB using various tools such as **Mongo Shell**, **Mongo Compass** and **PyMongo** (Python library).

1. Overview:

MongoDB is an open-source, cross-platform, document-oriented NoSQL (Not only SQL) database system developed in C++ programming language. MongoDB is designed to handle large volume of structured and unstructured data that will be stored in BSON format *(binary style of JSON documents),* which makes this database system extremely flexible and scalable with high performance and high available. MongoDB also provides driver support for most of the popular languages such as C, C#, C++, Go, Java, JavaScript, PHP, Python, Ruby, Rust, Scala, and Swift.

MongoDB Atlas is a cloud database solution that can be used by applications available globally and it offers fully managed service for MongoDB deployments across various cloud platforms such as AWS, Google Cloud, and Azure. **MongoDB Cloud** is a unified data platform that includes a global cloud database, search, data lake, mobile, and application services.

MongoDB was initially developed as PaaS (Platform as a Service) in 2007 by a New York based organization named **10gen** and later in 2009, it was introduced as an open source database server that is maintained and supported by **MongoDB Inc** *(**10gen** company name was renamed as **MongoDB Inc** in 2013).*

Unlike relational databases where data is stored in tables and columns, MongoDB follows document-oriented data storage using the concept of **Collections** (equivalent to tables in RDBMS) and **Documents** (equivalent to records in RDBMS). MongoDB documents consist of key-value pairs which hold the actual data in JSON format. A group of documents are clubbed together into the collection and all collections are stored in databases. In summary, MongoDB databases are used to store one or more collections of documents that contain data.

It is important to understand that MongoDB is not a relational database and is different over RDBMS:

- MongoDB is a non-relational, document-oriented database management system and works on document-based database where as RDBMS is a Relational database management system and works on relational databases.
- MongoDB database consists of collections that contain documents with a list of key-value pairs which has data while RDBMS database consists of tables that contain columns and rows which has data.
- In MongoDB, the data is stored in the form of documents. In RDBMS, a table row represents a single structured data item.

- MongoDB follows schema-less structure where one collection in a database can hold different structure of documents. RBDMS follows schema structure where a prior definition of schema is needed for tables in the database.
- MongoDB supports MongoDB Query Language (MQL) of BSON type while RDBMS supports Structured Query Language (SQL).
- In MongoDB, the database can be scaled both horizontally and vertically. In RDBMS, the database can be scaled vertically.
- MongoDB does not support complex join operations while RDBMS supports complex joins.
- MongoDB is optimized for write performance while RBDMS is optimized for high performance joins across multiple tables. MongoDB uses internal memory for storing working sets which makes it almost 100 times faster than traditional database systems.
- MongoDB follows the CAP theorem when RBDMS follows ACID properties.

MongoDB can be deployed with different topologies:

1. **Standalone** – In Standalone mode, a single MongoDB instance is deployed on a single host which takes care of all activities. A standalone MongoDB instance is ideal for development and testing purposes but not a good choice for a Production deployment due to its single point of failure.

2. **Replica Set –** A replica set (also known as cluster) consists of multiple interconnected MongoDB instances (`mongod` process) such that each instance run on a separate server or container. These instances can play a **Primary** role or **Secondary** role or **Arbitrary** role. Replica set architecture has automatic failover mechanism to ensure a new primary node is elected if the current primary node fails and provides high availability and data redundancy.

3. **Sharded Cluster –** A sharded cluster is a special type of cluster which provides data redundancy and high availability similar to replica set cluster but it also distributes data across shards that are hosted on multiple servers providing horizontal scalability. A **shard** contains a subset of data set so that multiple shards together makeup the entire data set for the cluster. Sharding helps to distribute the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster operations. Sharded cluster architecture consists of key components such as **Driver, Query Router** *(also called as `mongos`)***, Config Server** and **Shard**.

MongoDB is fully compatible with Windows OS, offering seamless integration and support for Windows environments. Whether you are running MongoDB on a local machine for development purposes or deploying it in a production environment on Windows servers, MongoDB provides native tools and utilities optimized for the Windows platform.

2. Prerequisites:

The following prerequisites need to be installed before using MongoDB.

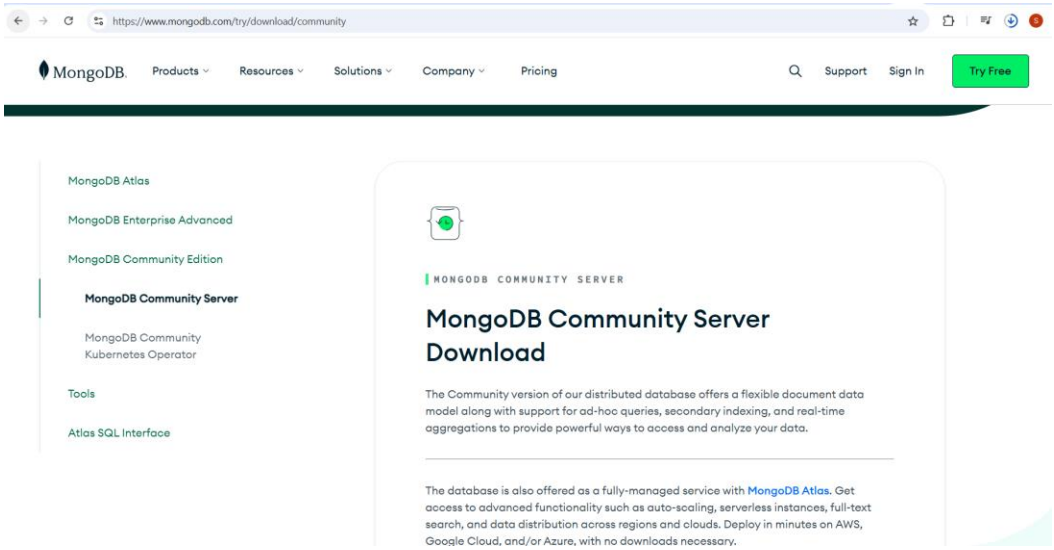
1. **File Archiver:** Any file archiver such as **7zip** or **WinRAR** is needed to unzip the downloaded Spark binaries. 7zip can be downloaded from the 7zip Downloads website and WinRAR can be downloaded from the RAR lab Downloads website.

3. Install Standalone MongoDB Instance:

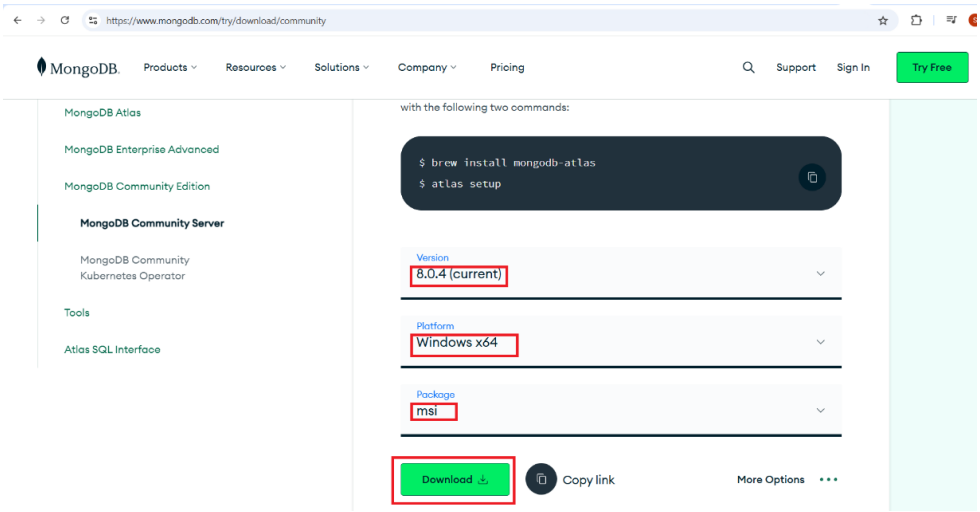
MongoDB is available in two server editions – **Community** and **Enterprise**. The **Community** edition includes all basic features of MongoDB while the **Enterprise** edition provides several additional features such as LDAP authentication, Kerberos authentication, and System Event Auditing but this edition is available for MongoDB Enterprise subscribers only.

In this tutorial, we will install MongoDB Community edition in Windows system. Follow the below installation steps for the same:

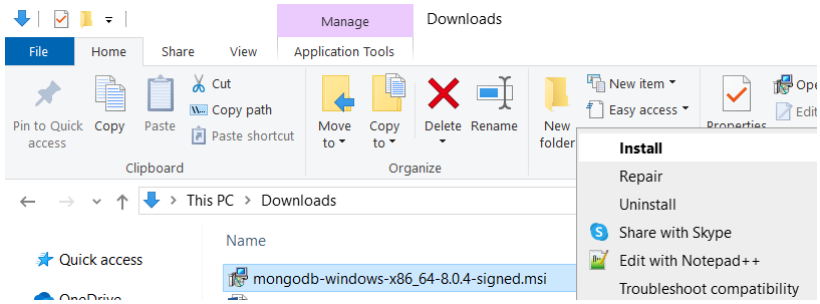
- Go to MongoDB Community Server Download Center page.



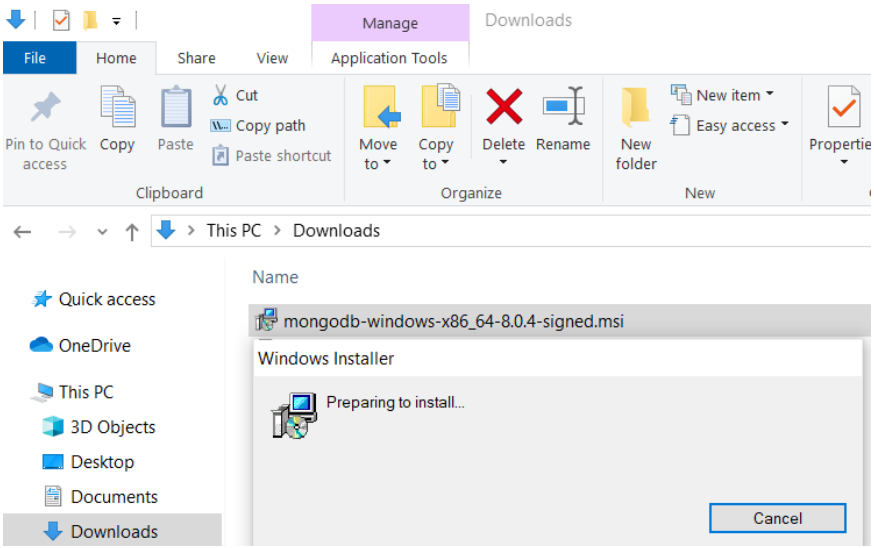
- Choose the latest **Version** *(at the time of writing this document, the latest version is 8.0.4)* and **Platform** as `Windows X64` and **Package** as `msi` and click on **Download** button which downloads the file into your **Downloads** folder in your machine.



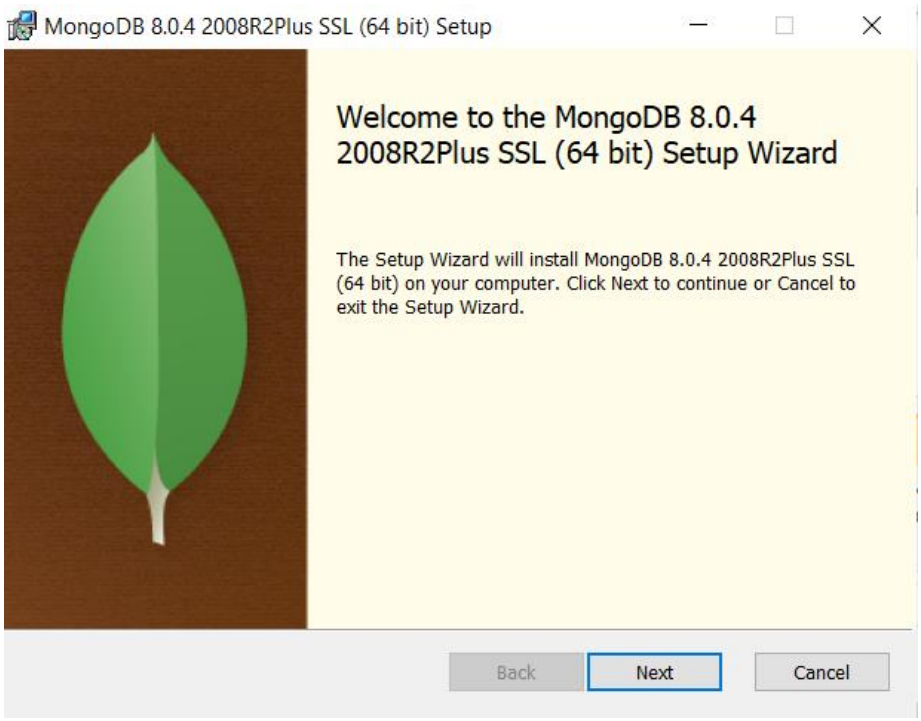
- Right click on the downloaded file `mongodb-windows-x86_64-8.0.4-signed.msi` in your **Downloads** folder and select **Install**.



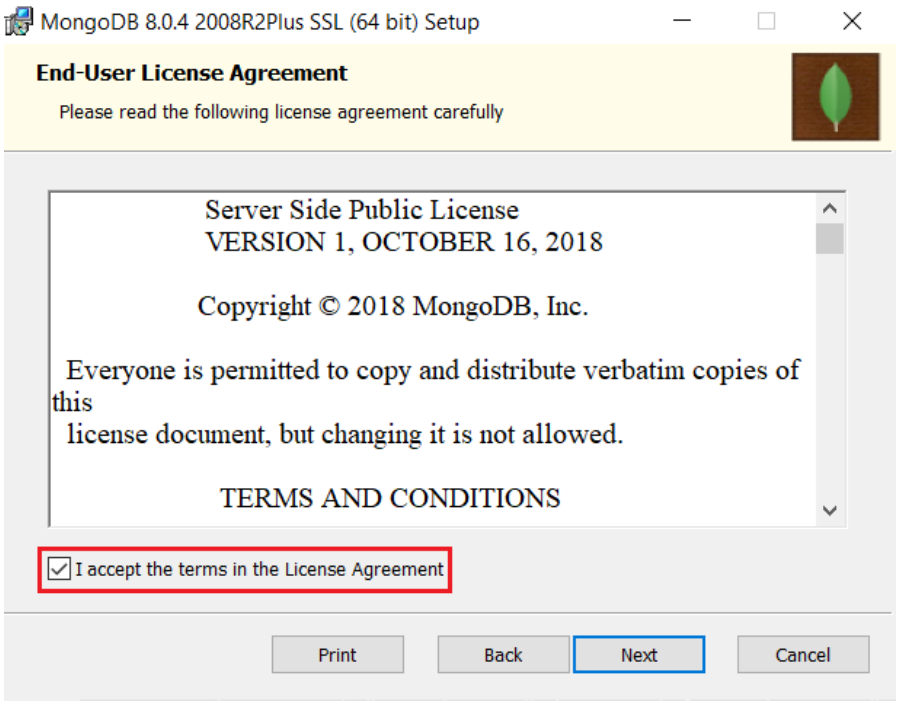
- It will start preparing to install.



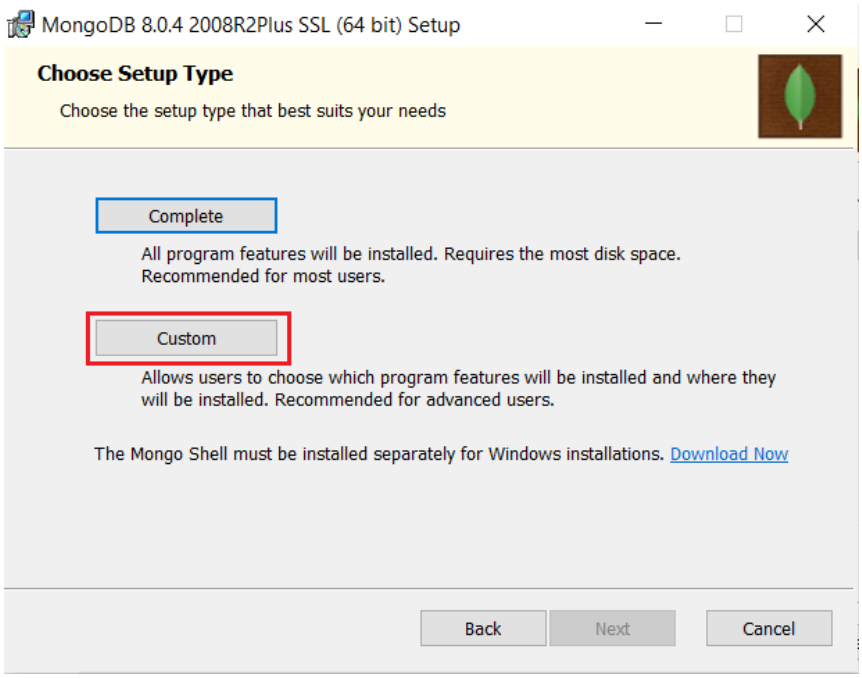
- It opens up a **MongoDB Setup** wizard where press **Next** to continue.



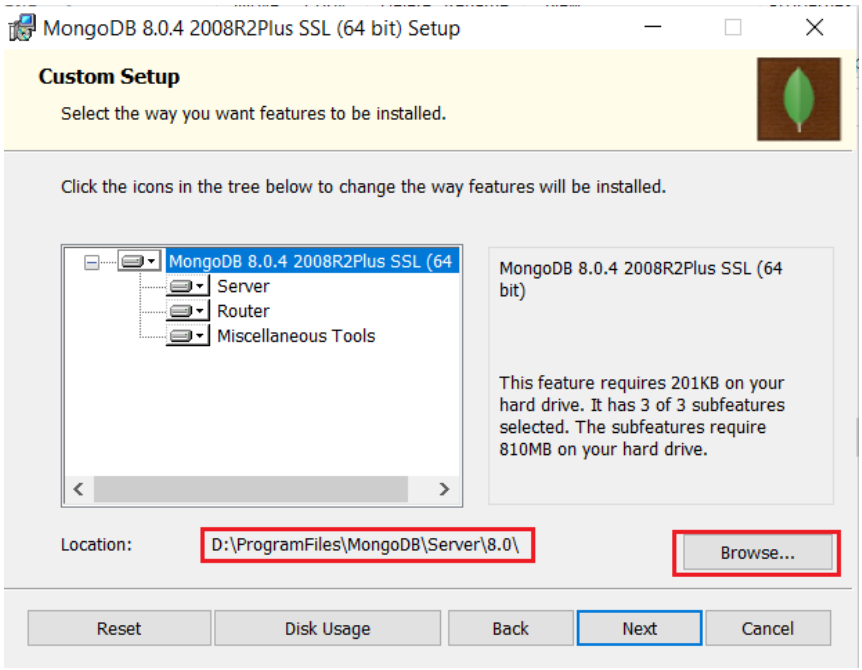
- Click on **I accept terms in the License Agreement** check box and press **Next**.



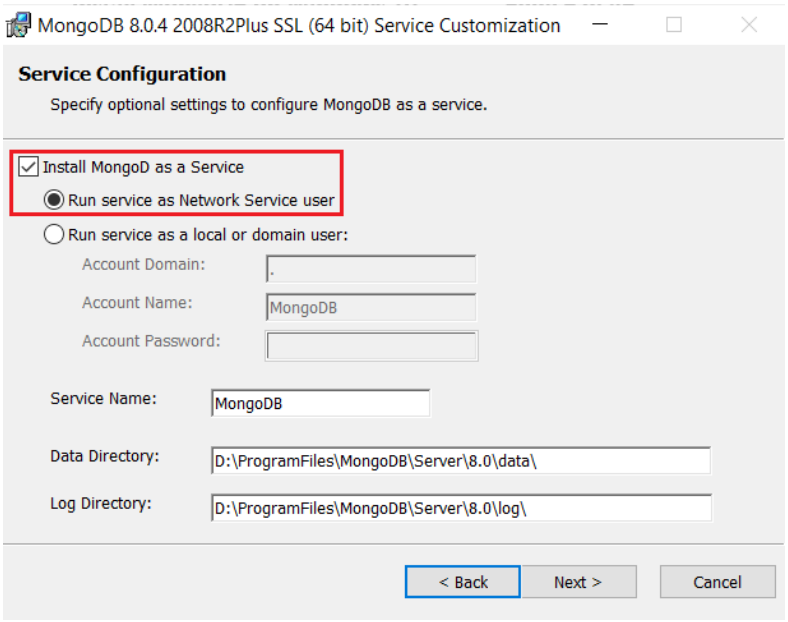
- On the **Setup Type** window, it provides two options **Complete** and **Custom**. While the **Complete** option installs all features in the default location, the **Custom** option allows us to install MongoDB in the custom path. So, press **Custom** button here.



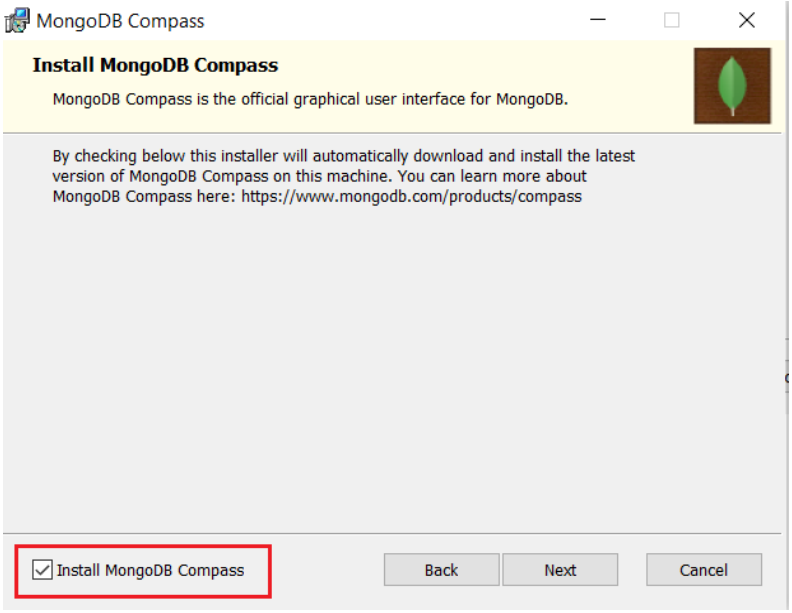
- On the **Custom Setup** window, click on **Browse** and choose the location where you want to install it. Here, I am choosing my install path as `D:\ProgramFiles\MongoDB\Server\8.0` and press **Next**.



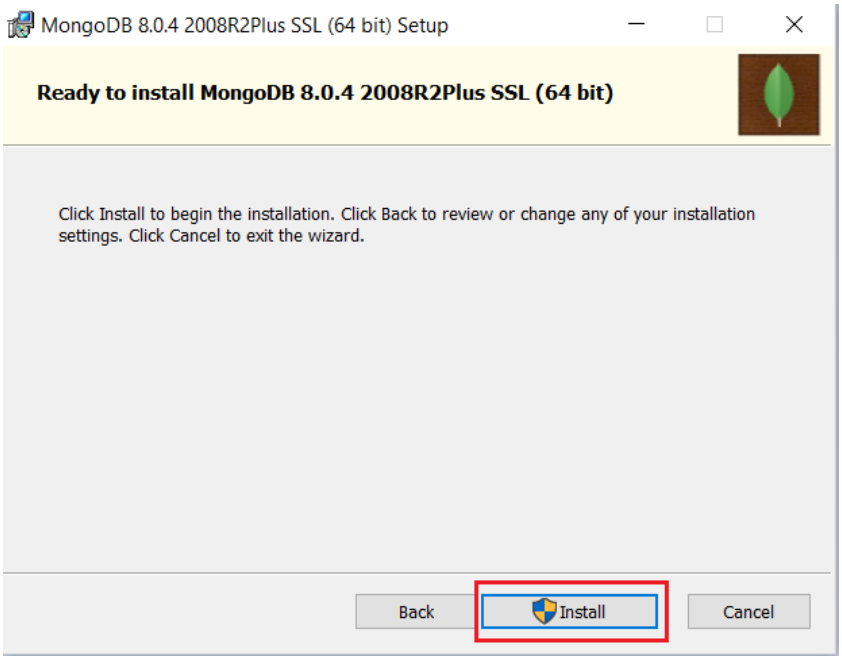
- Since we want to run the MongoDB as a service, keep the option of installing MongoDB as a service and let it run as Network Service user. If you want to run the service as a local user, then select **Run service as a local or domain user** option and provide your Account details and press **Next**.



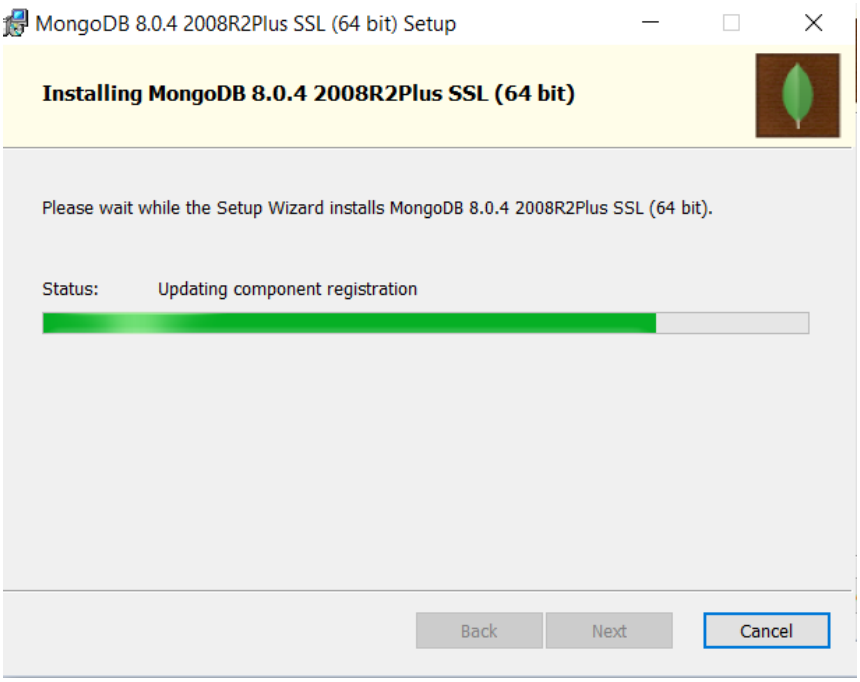
- Select **Install MongoDB Compass** checkbox to install the latest version of MongoDB Compass application *(**MongoDB Compass** is a GUI tool provided to connect and execute queries against MongoDB database).* Press **Next** to continue.



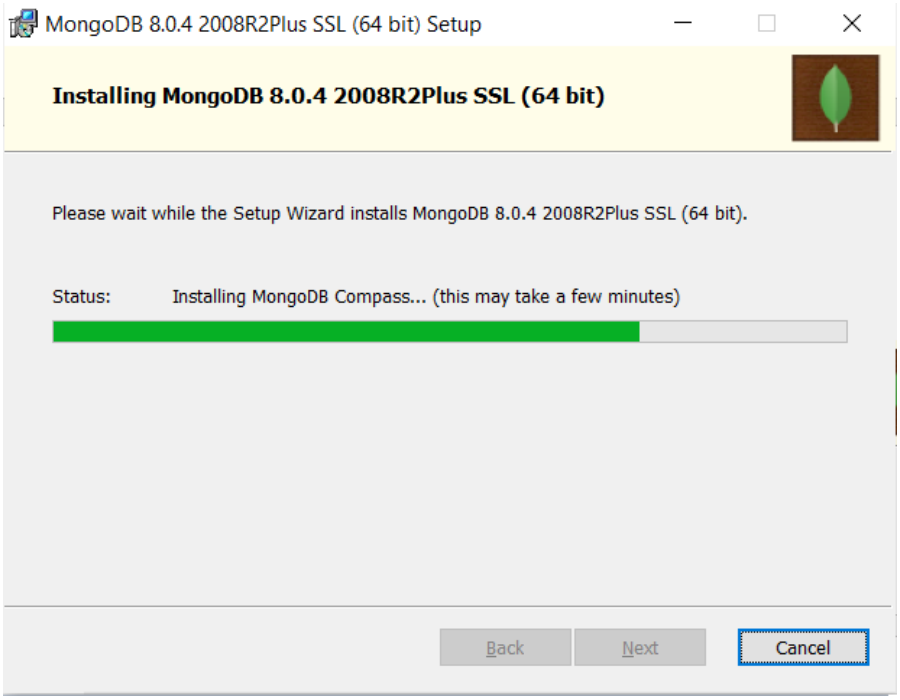
- Now, we are ready to install. Press **Install** button to continue



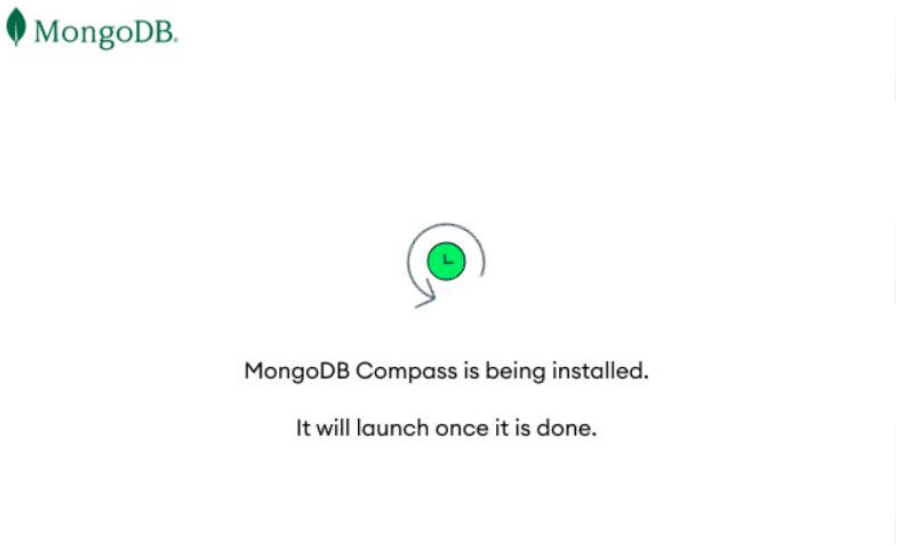
- On the setup wizard, you can see that it starts with installing MongoDB components and registering them.



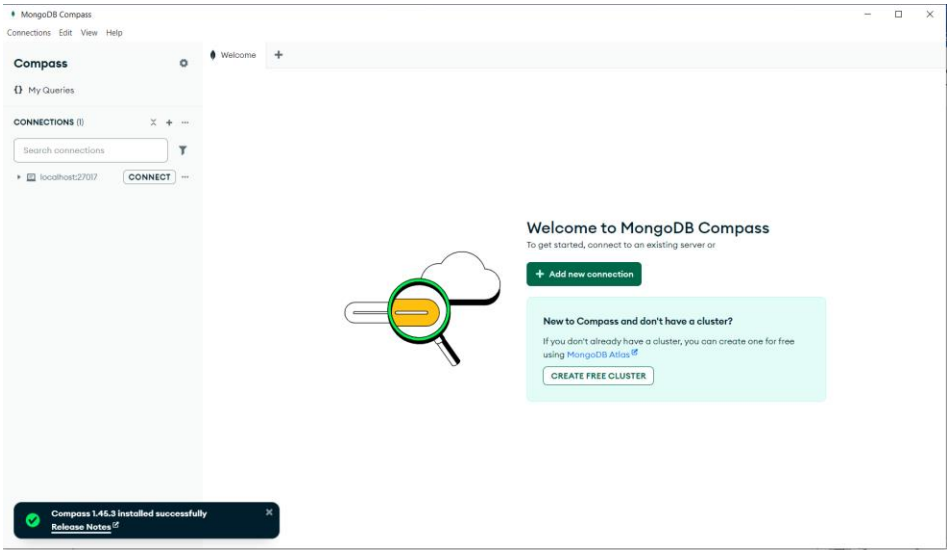
- It continues with installing MongoDB Compass application which may take a few minutes to complete.



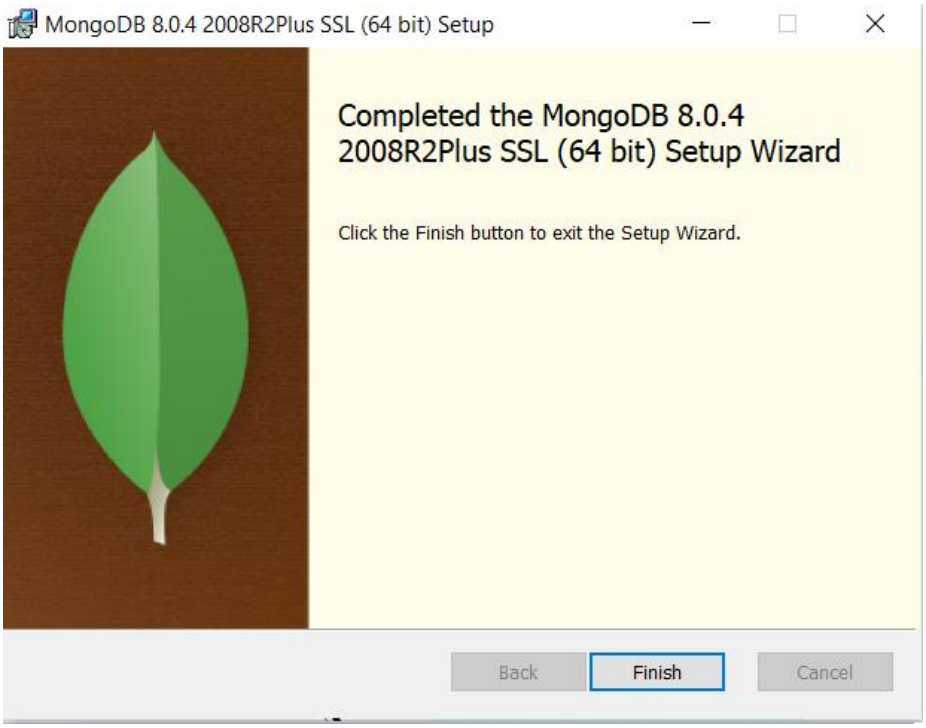
- While this installation is in progress, it opens up an application where you can see a message that **MongoDB compass is being installed**.



- After the installation is complete, it launches **MongoDB Compass** application with connections list to connect.



- On the **MongoDB Setup** Wizard, click on **Finish** to exit.



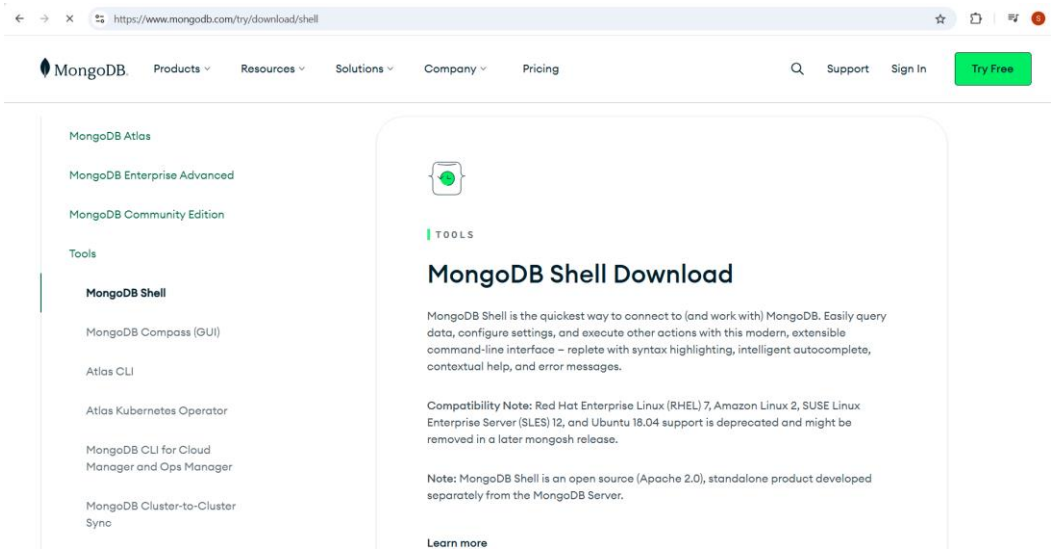
4. Install Standalone MongoDB Shell:

MongoDB Shell is a powerful command-based tool to interact with MongoDB databases. This tool is commonly used for:

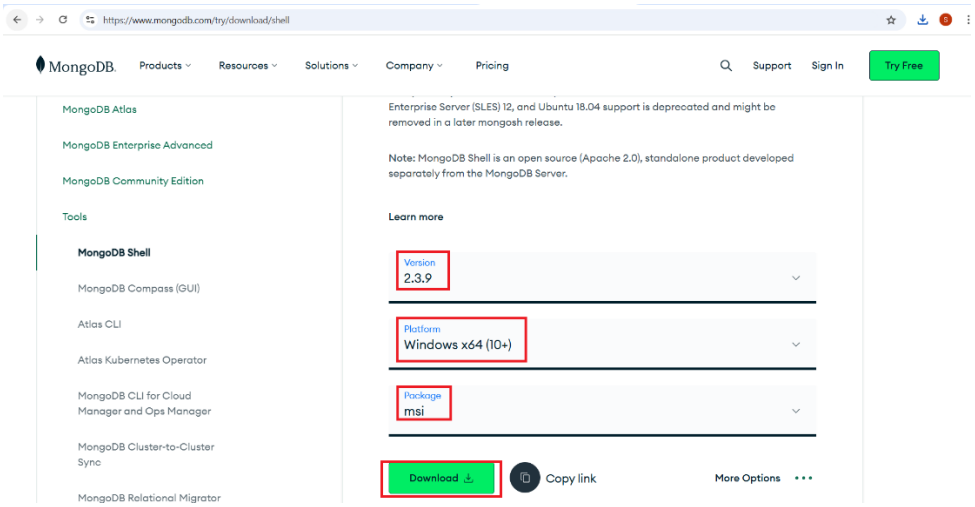
1. Database Administration *(to monitor and manage databases, backing up data, checking system status, or adjusting user permissions)*
2. Database Migration *(exporting, importing, manipulating data while moving data between different MongoDB instances or upgrading versions)*
3. Quick Querying and Troubleshooting *(run quick queries or perform data analysis, run diagnostic commands or examine logs)*
4. Automation and Scripting *(MongoDB shell scripts to automate repetitive tasks such as data cleanup, backups, or batch processing, helping improve productivity and reduce manual errors)*
5. Application Development *(to test queries, view documents, and interact with the database to ensure the application is working as expected)*

By default, the **MongoDB Shell** is not installed with MongoDB Server. Follow the below step by step procedure to install MongoDB Shell in Windows System.

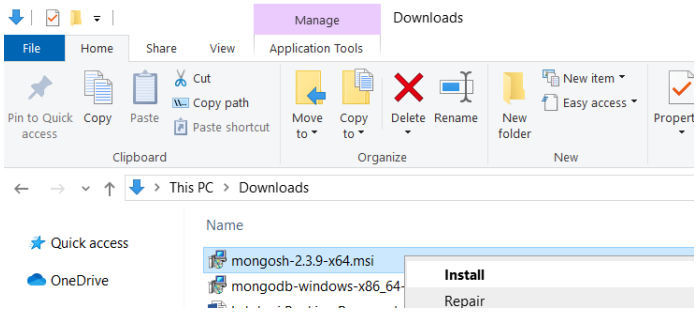
- Go to [MongoDB Shell Download](https://www.mongodb.com/try/download/shell) page.



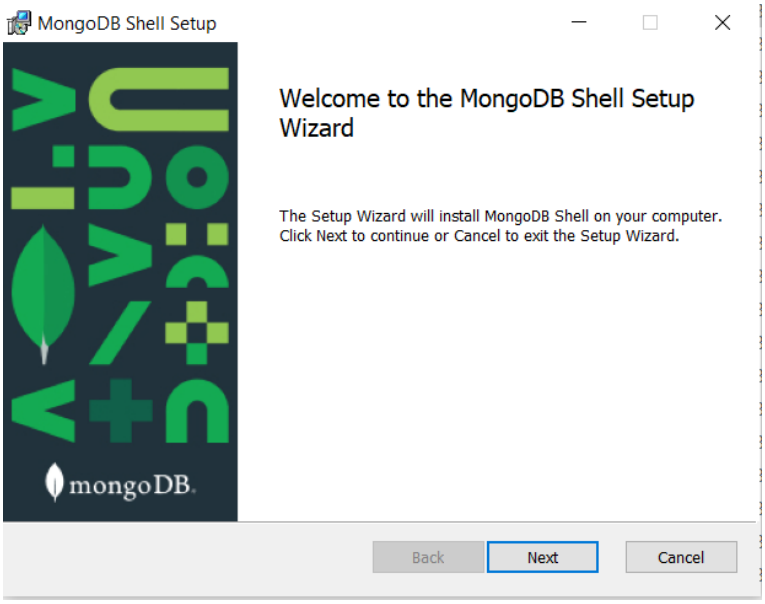
- Choose the latest **Version** *(at the time of writing this document, the latest version is 2.3.9)* and **Platform** as `Windows X64` and **Package** as `msi` and click on **Download** button which downloads the file into your **Downloads** folder in your machine.



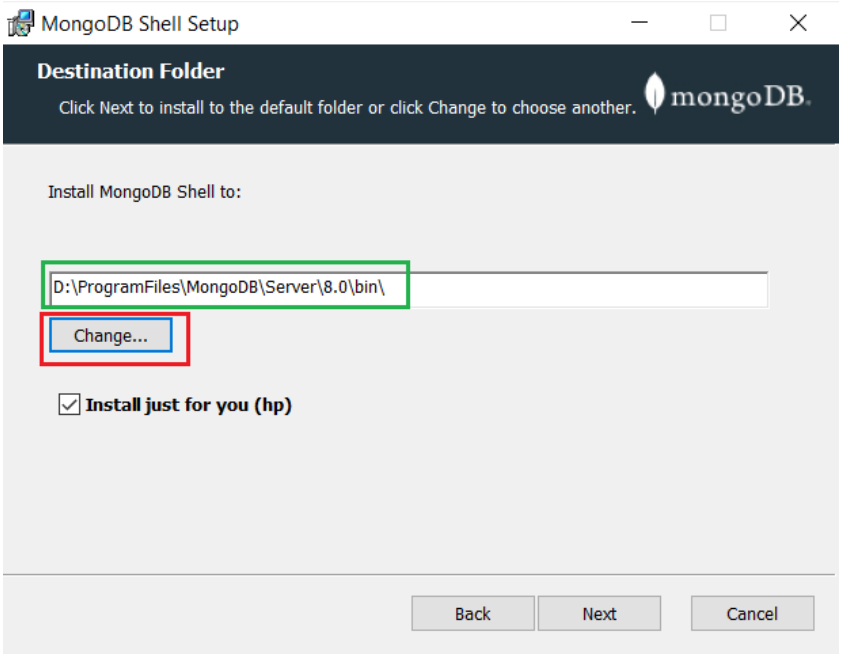
- Right click on the downloaded file `mongodb-2.3.9-x64.msi` in your **Downloads** folder and select **Install**.



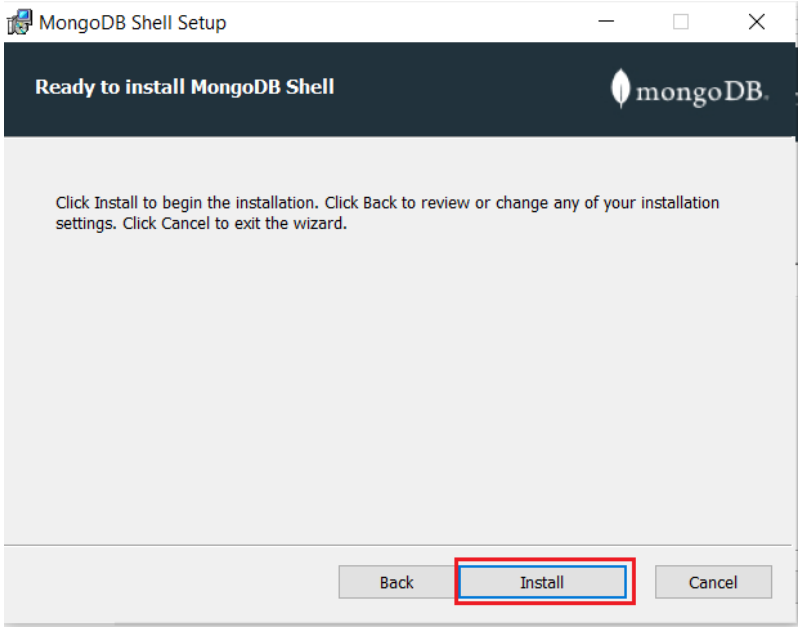
- It opens up a **MongoDB Setup wizard** where press **Next** to continue.



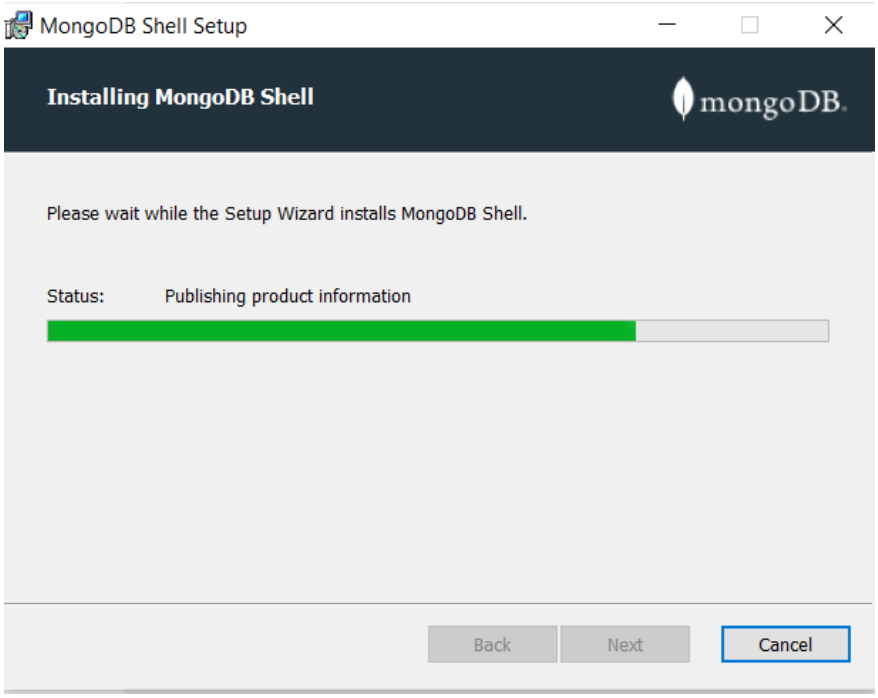
- On the **Destination Folder** window, click on **Change** and choose the location where you want to install it. By default, it tries to install at `C:\Users\hp\AppData\Local\Programs\mongosh` location but here, I am choosing my install path as `D:\ProgramFiles\MongoDB\Server\8.0\bin` and press **Next**.



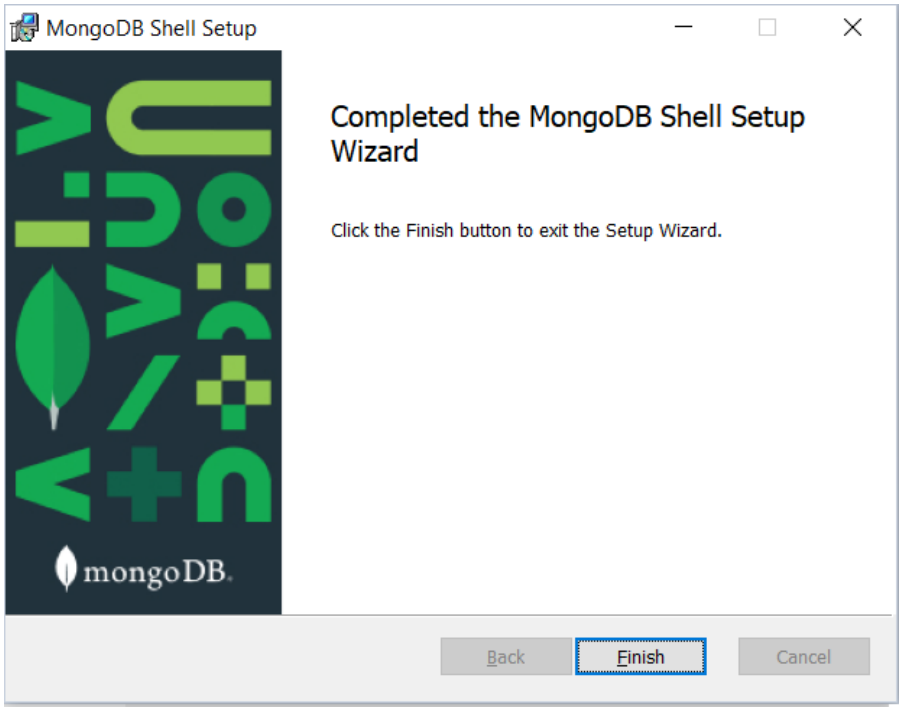
- Now, we are ready to install MongoDB Shell. Press **Install** button to continue.



- On the setup wizard, you can see the installation process.



- Click on **Finish** to exit the wizard.



5. Configure Standalone MongoDB:

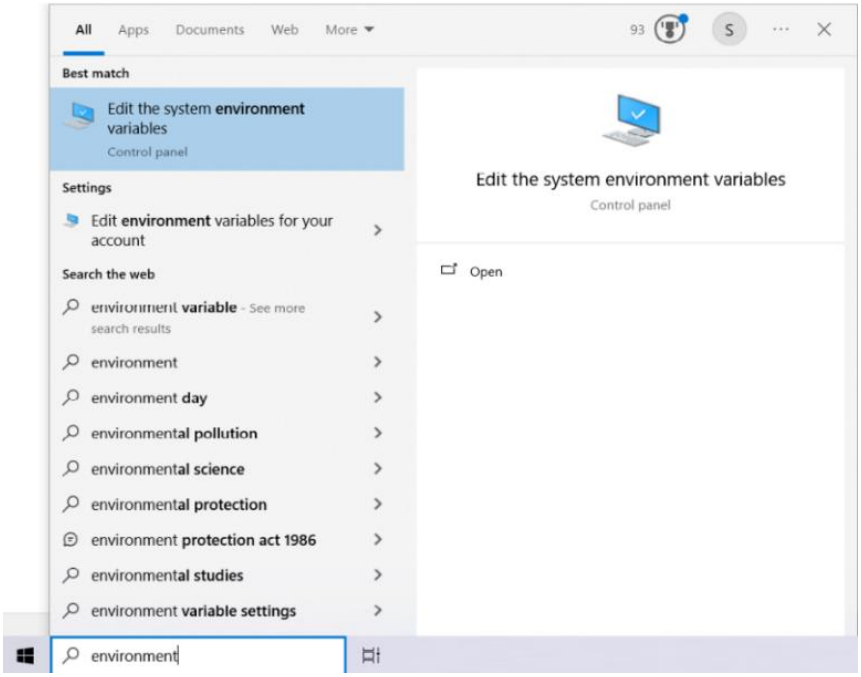
After the installation is complete, the next step is to configure the `PATH` environment variable defining the MongoDB `bin` location and verify the installation.

5.1. Setup Environment Variable:

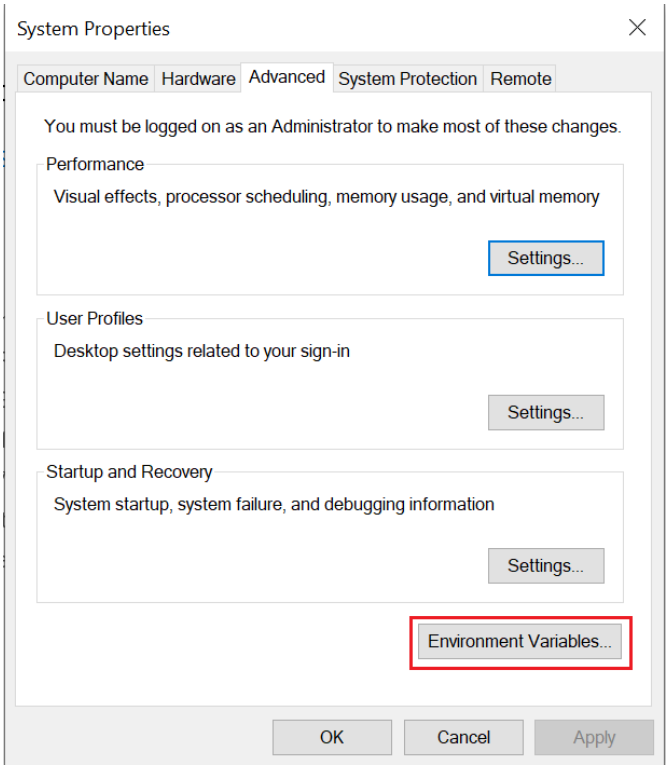
You can configure the `PATH` environment variable either under **User environment variables** or **System environment variables** depending on MongoDB configuration needed **for a single user** or **for multiple users**.

In this tutorial, we will update under User environment variables since we are configuring MongoDB for a single user. If you would like to configure for multiple users, then define System environment variables.

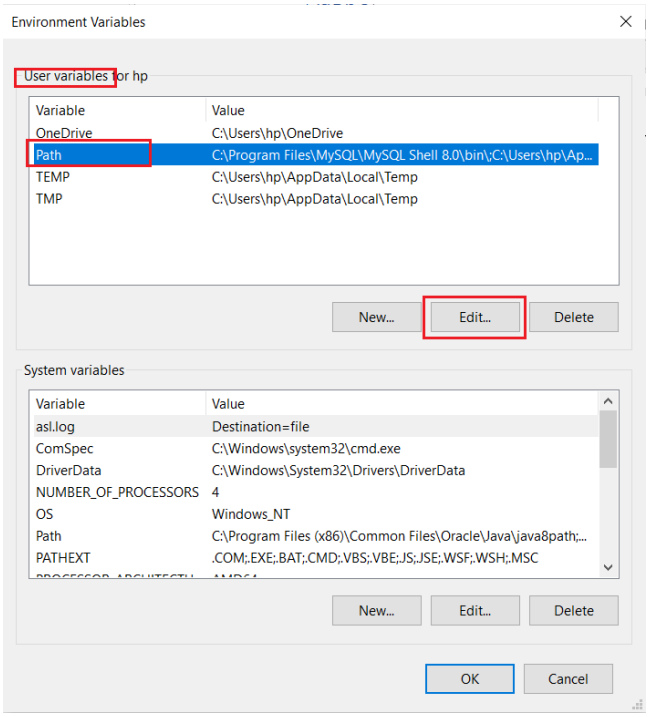
In the Windows search bar, start typing “environment variables” and select the first match which opens up **System Properties** dialog.



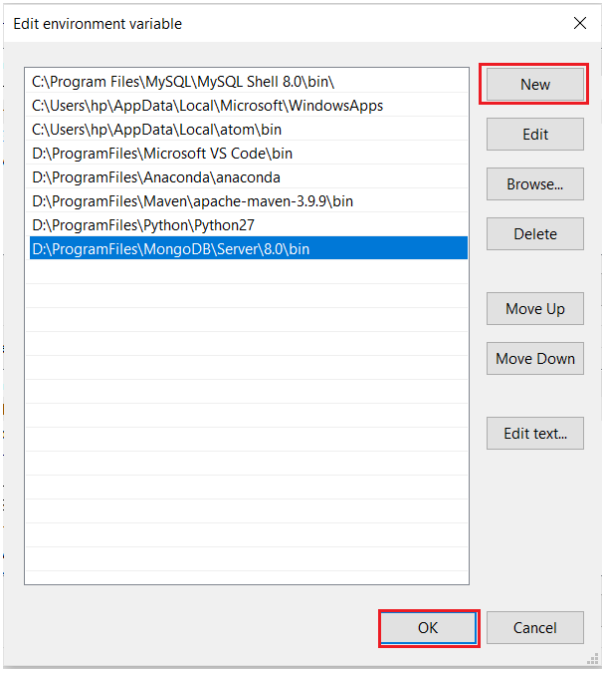
On the **System Properties** window, press **Environment Variables** button.



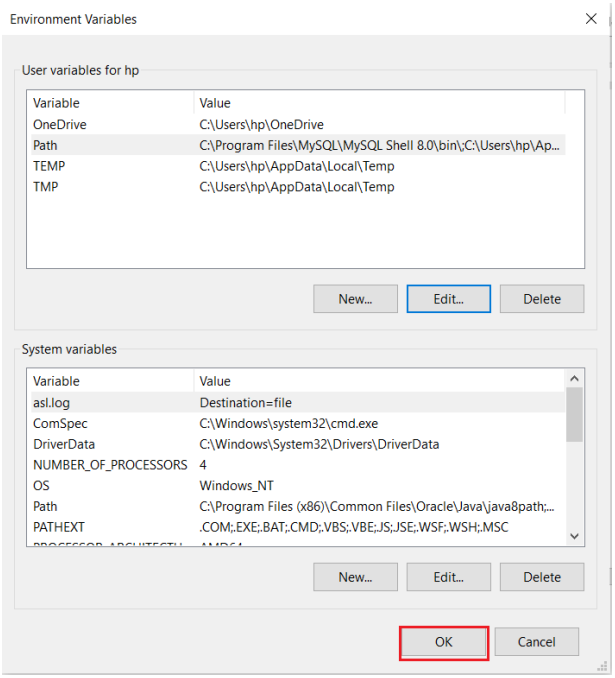
In the **Environment Variables** dialog, select `PATH` variable and press **Edit** button under **User variables** section.



Press **New** and add the following value *(note that this is the* `bin` *folder path of our MongoDB install location)* and press **OK**.
`D:\ProgramFiles\MongoDB\Server\8.0\bin`



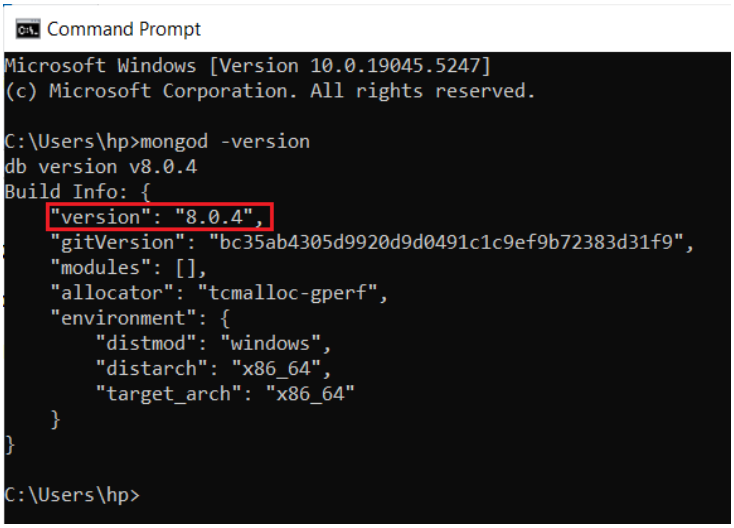
Press **OK** again to apply the environment variable changes and close window.



5.2. Verify MongoDB Installation:

Open **Command Prompt** and run the following command to verify if MongoDB is installed properly:

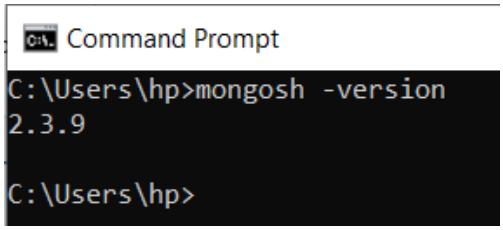
```
mongod -version
```



Here, you can see the **MongoDB version 8.0.4** was installed

Run the following command to verify if MongoDB Shell is installed properly:

```
mongosh --version
```



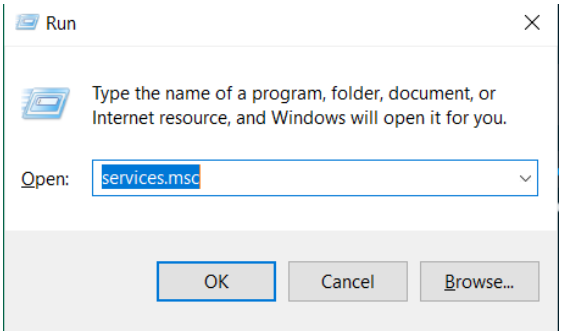
Here, you can see **2.3.9** version was installed.

6. Start Standalone MongoDB:

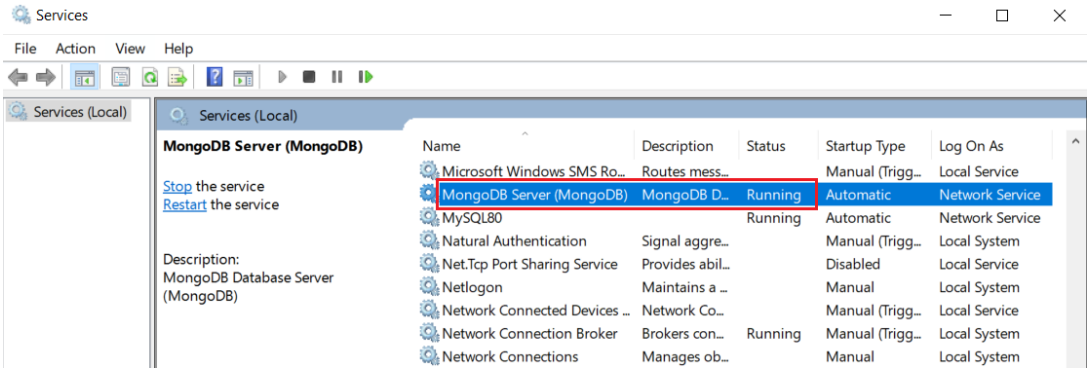
It is time to start the MongoDB service and test the connection.

6.1. Start MongoDB Service:

Since we installed MongoDB as a service, it would have started and running already. To verify this, press **Windows +R** in your keyboard system to open **Run** application and type `services.msc` and press OK.

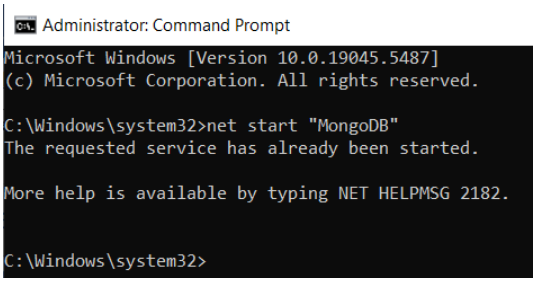


On the **Services** application, look for **MongoDB Server** service and review the **Status** which shows **Running**. If not Running already, click on **Start** link to start the service.



You can also start the MongoDB service through **Command Prompt** application as **Administrator** using the below command:

```
net start "MongoDB"
```



6.2. Verify MongoDB Status:

Open **Command Prompt** application and execute the below command:

```
sc query "MongoDB"
```

```
C:\Users\hp>sc query "MongoDB"

SERVICE_NAME: MongoDB
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_PRESHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

C:\Users\hp>
```

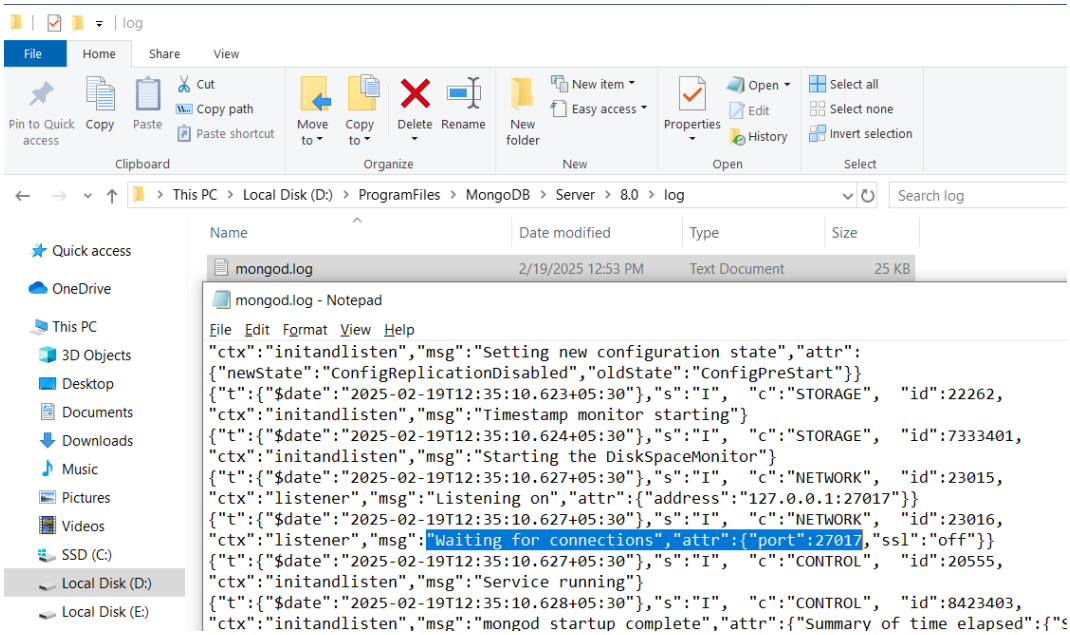
Here, you can see the STATE as **RUNNING**.

You can also verify if the MongoDB port (`27017`, by default) is listening by running the following command:

```
netstat -an | find "27017"
```

```
C:\Users\hp>netstat -an | find "27017"
  TCP    127.0.0.1:27017        0.0.0.0:0              LISTENING

C:\Users\hp>
```

Additionally, you can open `mongod.log` file located at `D:\ProgramFiles\MongoDB\Server\8.0\log` location and look for **Waiting for connections on port 27017** message which indicates that MongoDB service has been started and running successfully.



7. MongoDB Shell:

MongoDB provides an interactive tool called `mongosh` which is a JavaScript environment to communicate with MongoDB database and execute queries using `mongosh` commands.

Using `mongosh`, you can connect to a MongoDB instance running either locally or remotely.

a. Connect to Local Instance:

Open **Command prompt** and simply run the below command to connect to a MongoDB instance locally.

```
mongosh
```

`

This command will connect to the MongoDB instance running in `localhost` on default port `27017`. This method is also equivalent to connecting with a connection string as below:

```
mongosh "mongodb://localhost:27017"
```

If you need to connect to a specific database (let's say `admindb`), you will have to add the name of the database in the connection string as below:

```
mongosh "mongodb://localhost:27017/admindb"
```

If you want to connect to a specific database (let's say `admindb`) without connection string, just provide the database name in the `mongosh` command:

```
mongosh admindb
```

If your MongoDB instance is running on a custom port (let's say `28089`), you can use `--port` command line option to connect:

```
mongosh --port 28089
```

b. Connect to Remote Instance:

You can connect to a MongoDB instance running remotely using connection string as below:

```
mongosh "mongodb://mongodb0.example.com:29058"
```

In the above command, `mongodb0.example.com` is the remote host name and `29058` is the port where MongoDB instance is running.

You can also use command-line options to make a remote connection:

```
mongosh --host mongodb0.example.com --port 29058
```

You can connect to a MongoDB replica set using connection string as below:

```
mongodb://username:password@db0.example.com:27017,db1.example.com:270
17,db2.example.com:27017/?replicaSet=myRepl
```

In the above command, `myRepl` is the replica set name that has 3 members with host names `db0.example.com, db1.example.com` and `db2.example.com` where MongoDB instances are running on `29017` port.

If you need to connect to MongoDB installed on cloud such as Mongo Atlas, use the below syntax:

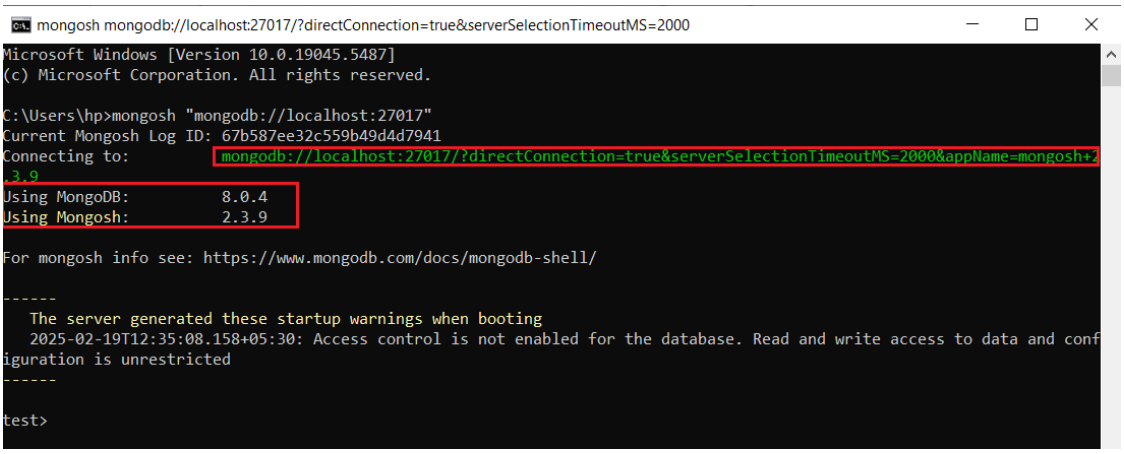
```
mongodb+srv://[username:password@]host:port[/[defaultauthdb][?options
]]
```

For example, use the below command to connect to `myRepl` replica set installed on host `mongodb0.example.com` with authorization source as `admin`:

```
mongodb+srv://username:password@mongodb0.example.com/?authSource=admi
n&replicaSet=myRepl
```

For now, open **Command Prompt** and run the below command to connect to our local MongoDB instance:

```
mongosh "mongodb://localhost:27017"
```



When you are connected successfully, you can see the fully qualified connection string of the MongoDB instance and the installed versions of MongoDB, Mongosh on the window.
Note that by default, Mongosh connects to the `test` database *(though this database is not available)* and hence give you `test>` prompt to proceed with executing queries.

Now, let us execute some mongosh commands for your awareness.

7.1. Get Mongo:

Use the below mongosh command to get the current Mongo instance connection URL.

```
db.getMongo()
```

```
test> db.getMongo()
mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.9
test>
```

7.2. Get Version:

Use the below mongosh command to get the installed version of MongoDB instance.

```
db.version()
```

```
test> db.version()
8.0.4
test>
```

7.3. Get Help:

Use `help()` command to get the list of commands that can be executed on MongoDB shell:

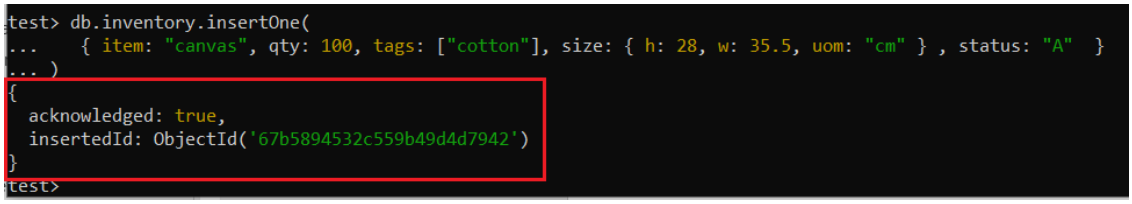
```
db.help()
```

```
test> db.help()

  Database Class:

    getMongo                                Returns the current database connection
    getName                                 Returns the name of the DB
    getCollectionNames                      Returns an array containing the names of all collections in the current d
atabase.
    getCollectionInfos                      Returns an array of documents with collection information, i.e. collectio
n name and options, for the current database.
    runCommand                              Runs an arbitrary command on the database.
    adminCommand                            Runs an arbitrary command against the admin database.
    aggregate                               Runs a specified admin/diagnostic pipeline which does not require an unde
rlying collection.
    getSiblingDB                            Returns another database without modifying the db variable in the shell e
nvironment.
    getCollection                           Returns a collection or a view object that is functionally equivalent to
using the db.<collectionName>.
    dropDatabase                            Removes the current database, deleting the associated data files.
    createUser                              Creates a new user for the database on which the method is run. db.create
User() returns a duplicate user error if the user already exists on the database.
    updateUser                              Updates the user's profile on the database on which you run the method. A
n update to a field completely replaces the previous field's values. This includes updates to the user's roles array.
    changeUserPassword                      Updates a user's password. Run the method in the database where the user
is defined, i.e. the database you created the user.
    logout                                  Ends the current authentication session. This function has no effect if t
he current session is not authenticated.
    dropUser                                Removes the user from the current database.
    dropAllUsers                            Removes all users from the current database.
```

7.4. Insert Documents:

Run the following `mongosh` command to insert one document into a collection named `inventory` in the current database. The document data must be provided in JSON format with key-value pairs. If the given collection does not exist already in the database, the insert operation creates the collection and add a document into it.

```
db.inventory.insertOne(
   { item: "canvas", qty: 100, tags: ["cotton"], size: { h: 28, w:
35.5, uom: "cm" }, status: "A"  }
)
```



If the document insertion is successful, the `insertOne()` method returns the `acknowledged` as `true` and provides the newly inserted document's `objectId` i.e. `_id` field value.

Run the following command to insert multiple documents into a collection:

```
db.inventory.insertMany([
   { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14,
w: 21, uom: "cm" }, status: "A" },
   { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5,
uom: "cm" }, status: "P" },
   { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19,
w: 22.85, uom: "cm" }, status: "D" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" },
status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" },
status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" },
status: "A" },
])
```

```
test> db.inventory.insertMany([
...    { item: "journal", qty: 25, tags: ["blank", "red"], size: { h: 14, w: 21, uom: "cm" } , status: "A" },
...    { item: "mat", qty: 85, tags: ["gray"], size: { h: 27.9, w: 35.5, uom: "cm" } , status: "P" },
...    { item: "mousepad", qty: 25, tags: ["gel", "blue"], size: { h: 19, w: 22.85, uom: "cm" } , status: "D" },
...    { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
...    { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
...    { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" },
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('67b58a3132c559b49d4d7943'),
    '1': ObjectId('67b58a3132c559b49d4d7944'),
    '2': ObjectId('67b58a3132c559b49d4d7945'),
    '3': ObjectId('67b58a3132c559b49d4d7946'),
    '4': ObjectId('67b58a3132c559b49d4d7947'),
    '5': ObjectId('67b58a3132c559b49d4d7948')
  }
}
test>
```

When all documents insertion is successful, the `insertMany()` method returns the `acknowledged` as `true` and provides the newly inserted documents `_id` field values.

7.5. Search Documents:

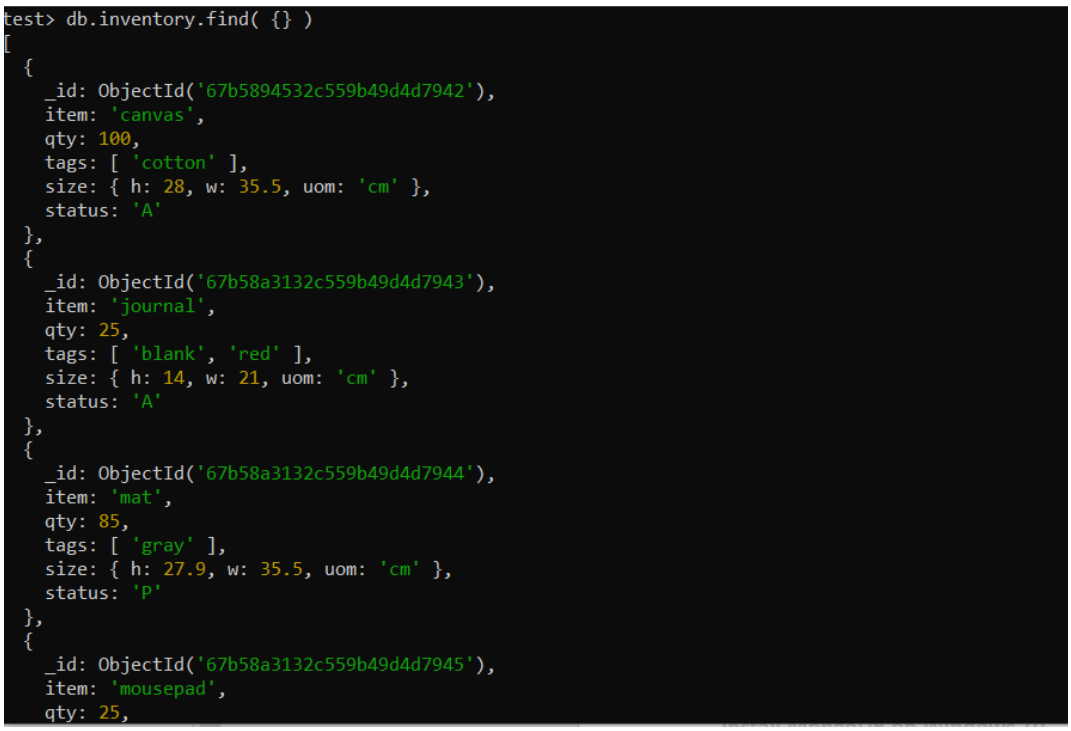
Run the following command to find a document that consists of `item: "canvas"` field in the `inventory` collection. If the collection consists of multiple documents which have `item: "canvas"`, then `find()` method retrieves the first record found.

```
db.inventory.find( { item: "canvas" } )
```

```
test> db.inventory.find( { item: "canvas" } )
[
  {
    _id: ObjectId('67b5894532c559b49d4d7942'),
    item: 'canvas',
    qty: 100,
    tags: [ 'cotton' ],
    size: { h: 28, w: 35.5, uom: 'cm' },
    status: 'A'
  }
]
test>
```

Run the following command to retrieve all documents from the `inventory` collection:

```
db.inventory.find( {} )
```



7.6. Update Documents:

To update a document, MongoDB provides many update operators, such as `$set`, to modify field values. Refer [MongoDB documentation] to know more about update operators.

Use the following syntax while using update methods:

```
{
  <update operator>: { <field1>: <value1>, ... },
  <update operator>: { <field2>: <value2>, ... },
  ...
}
```

Run the following command to update the first document where `item` equals "`paper`":

```
db.inventory.updateOne(
   { item: "paper" },
   {
     $set: { "size.uom": "cm", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

This update operation:

- uses the `$set` operator to update the value of the `size.uom` field to `"cm"` and the value of the `status` field to `"P"`,
- uses the `$currentDate` operator to update the value of the `lastModified` field to the current date. If `lastModified` field does not exist, `$currentDate` will create the field.
- returns the `matchedCount` and `modifiedCount` to understand how many documents it was able to match with the given condition and were updated.

```
test> db.inventory.updateOne(
...     { item: "paper" },
...     {
...       $set: { "size.uom": "cm", status: "P" },
...       $currentDate: { lastModified: true }
...     }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test>
```

Run the following command to update all documents where `qty` less than `50`:

```
db.inventory.updateMany(
   { "qty": { $lt: 50 } },
   {
     $set: { "size.uom": "in", status: "P" },
     $currentDate: { lastModified: true }
   }
)
```

```
test> db.inventory.updateMany(
...     { "qty": { $lt: 50 } },
...     {
...       $set: { "size.uom": "in", status: "P" },
...       $currentDate: { lastModified: true }
...     }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
test>
```

Here, you can see that 3 documents have been matched and updated.

Mongosh also provides `replaceOne()` method to replace the entire content of a document except `_id` field.

Run the following command to replace the first document from the `inventory` collection where `item: "in"`:

```
db.inventory.replaceOne(
   { item: "paper" },
   { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
)
```

```
test> db.inventory.replaceOne(
...    { item: "paper" },
...    { item: "paper", instock: [ { warehouse: "A", qty: 60 }, { warehouse: "B", qty: 40 } ] }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
test>
```

7.7. Delete Documents:

We can delete one or more documents in a collection using `deleteOne()` or `deleteMany()` methods. We can also specify a criteria or filters that identify documents to delete.

Use the following syntax to specify a query filter while using delete operations:

```
{ <field1>: { <query_operator1>: <value1> }, ... }
```

In the above syntax, the query operator can be `$eq` (for equal to), `$gt` (for greater than), `$gte` (for greater than or equal to), `$lt` (for less than), `$lte` (for less than or equal to), `$ne` (for not equal to) etc. Refer MongoDB documentation to know more about query operators.

Run the following command to delete the first document from the `inventory` collection where `status` is `"D"`:

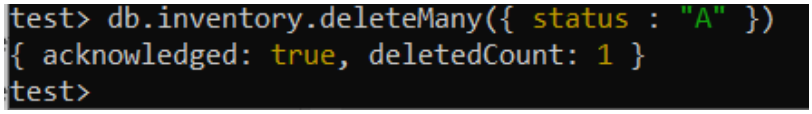
```
db.inventory.deleteOne( { status: "D" } )
```

```
test> db.inventory.deleteOne( { status: "D" } )
{ acknowledged: true, deletedCount: 1 }
test>
```

When the deletion of documents is successful, the `deleteOne()` method returns the `acknowledged` as `true` and provides the number of deleted documents.

Use the following command to delete all documents where `status` is `"A"`:

```
db.inventory.deleteMany({ status : "A" })
```

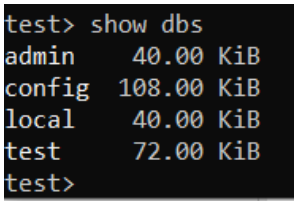


7.8. Create Database:

In MongoDB, databases are not created explicitly until some data is loaded into them. We can take the help of `use` command followed by a database name to switch to that database even if it does not exist.

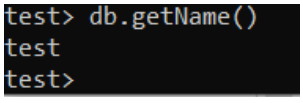
Run the below mongosh command to get the list of available databases in the current MongoDB instance:

```
show dbs
```



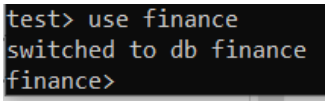
Run the below command to get the current database name:

```
db.getName()
```



Run the below mongosh command to switch to a database named `finance`.

```
use finance
```



7.9. Create Collection:

Use below command to explicitly create a collection in the current database:

```
db.createCollection("employee")
```

```
finance> db.createCollection("employee")
{ ok: 1 }
finance>
```

Use below command to get the list of available collections in the current database:

```
db.getCollectionNames()
```

```
finance> db.getCollectionNames()
[ 'employee' ]
finance>
```

You can also use the below command to fetch the available collections in the current database:

```
show collections
```

```
finance> show collections
employee
finance>
```

7.10. Drop Collection:

Use the following command to drop a collection named `employee`:

```
db.employee.drop()
```

```
finance> db.employee.drop()
true
finance>
```

7.11. Drop Database:

Use the below mongosh command to remove the current database.

```
db.dropDatabase()
```

```
finance> db.dropDatabase()
{ ok: 1, dropped: 'finance' }
finance>

finance> show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
test     72.00 KiB
finance>
```

To understand more about mongsh methods, refer the official [MongoDB documentation](#).

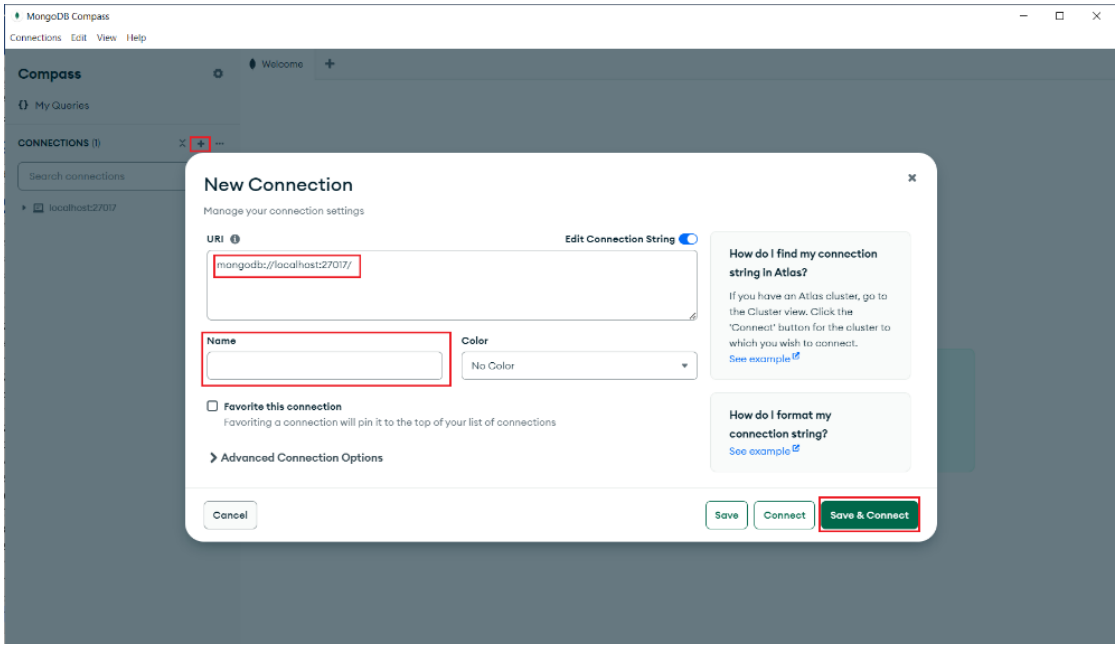
8. MongoDB Compass:

MongoDB Compass is a Graphical User Interface application provided for easy access to query, aggregate, and analyze Mongo data.

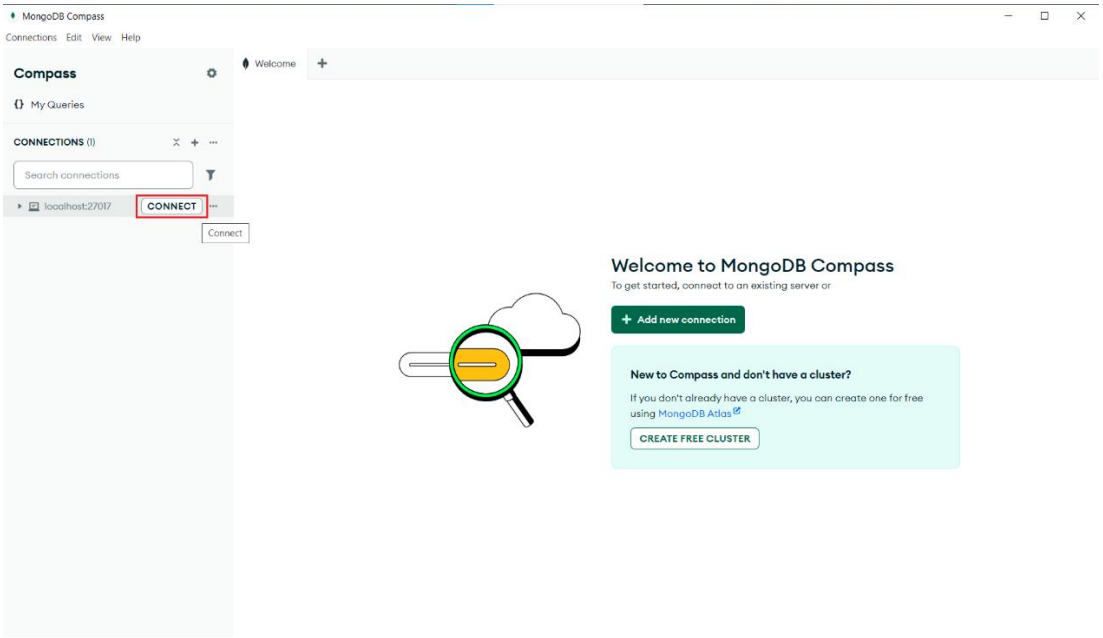
8.1. Connect Mongo:

Launch **MongoDB Compass** application in your Windows system. MongoDB Compass shows a default connection **localhost:27017** to connect to MongoDB instance running locally on default port 27017.

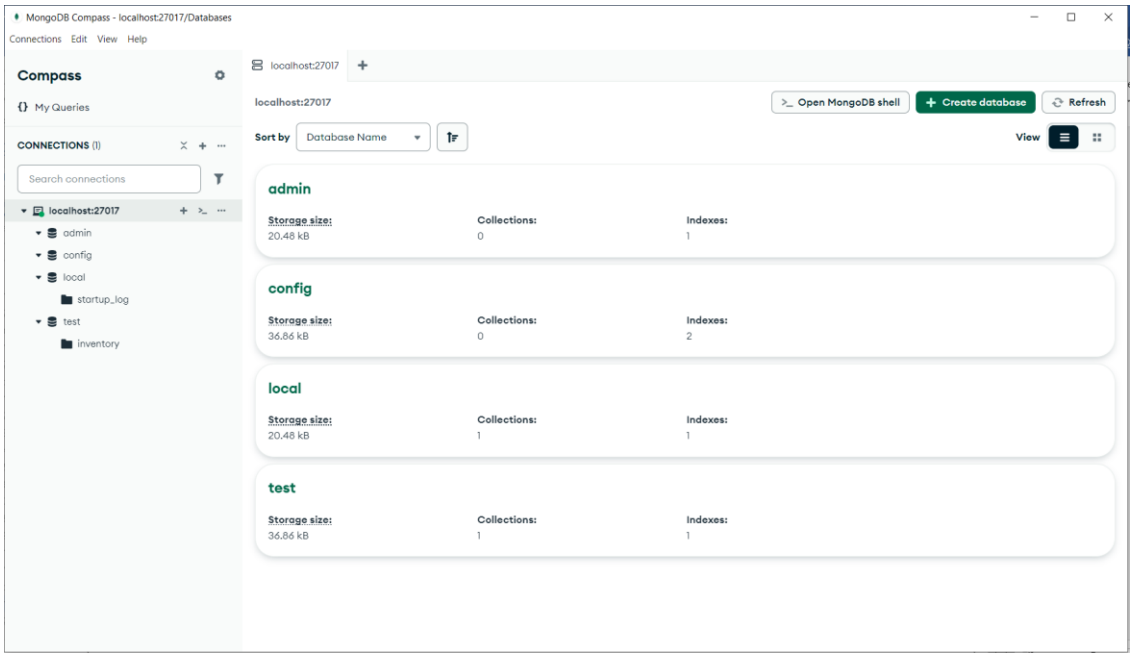
If you would like to connect to either a remote host or to MongoDB running on a different port, click on **+** icon next to **CONNECTIONS** which opens a **New Connection** window where you need to specify the connection string and name for reference and press **Save & Connect**.



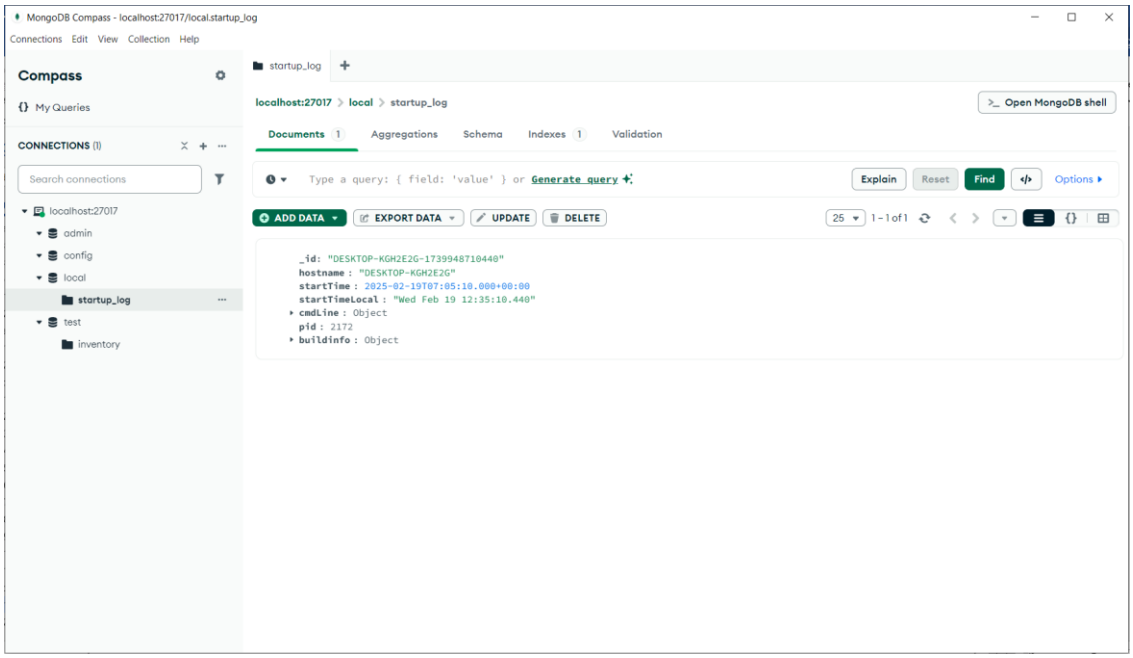
For now, let's click on **CONNECT** button next to **localhost:27017** to make a connection.



Once you are connected to the default localhost instance, you will see three default databases – `admin, config, local` and an additional database `test` that was created when we inserted documents into `inventory` collection through MongoDB Shell.



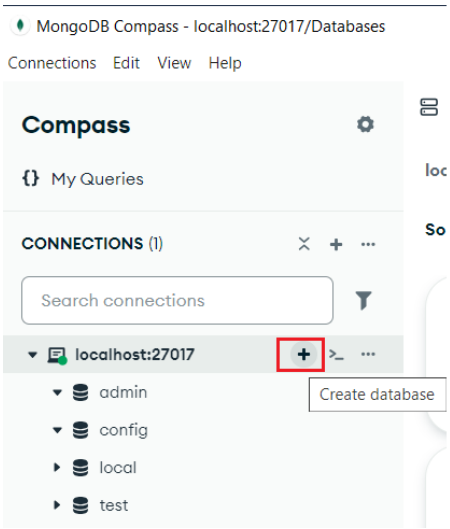
In the `local` database, you will see a collection named `startup_log` which consists of a document with list of key-value pairs which gives you the MongoDB startup information.



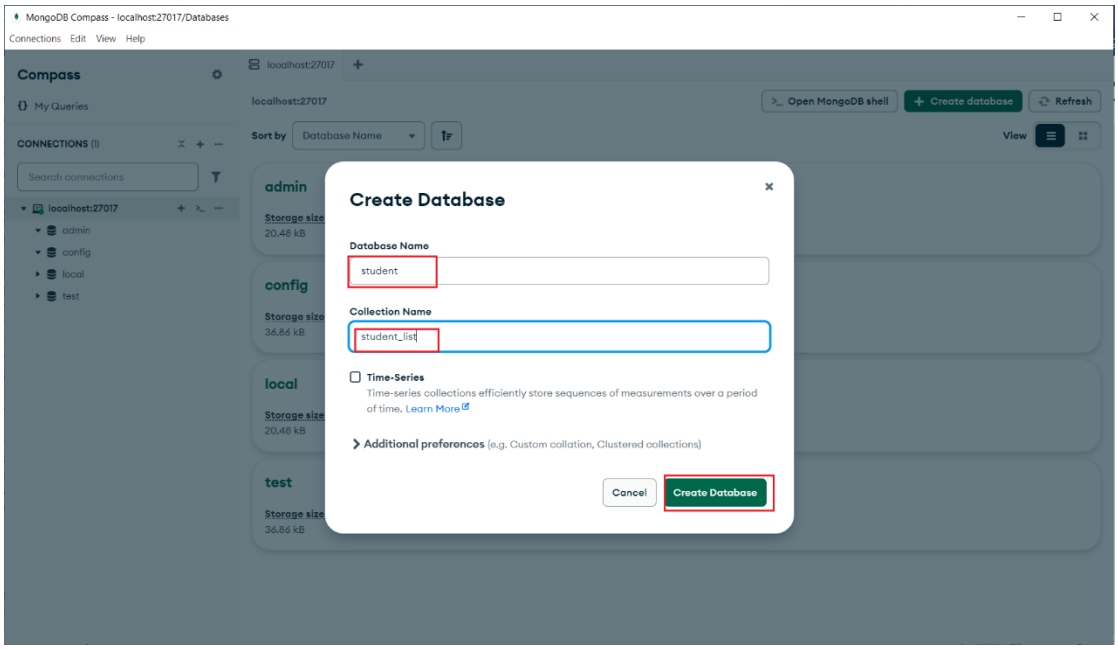
8.2. Create Database:

While creating a database from MongoDB Compass, it is necessary to specify the collection name without which database cannot be created.

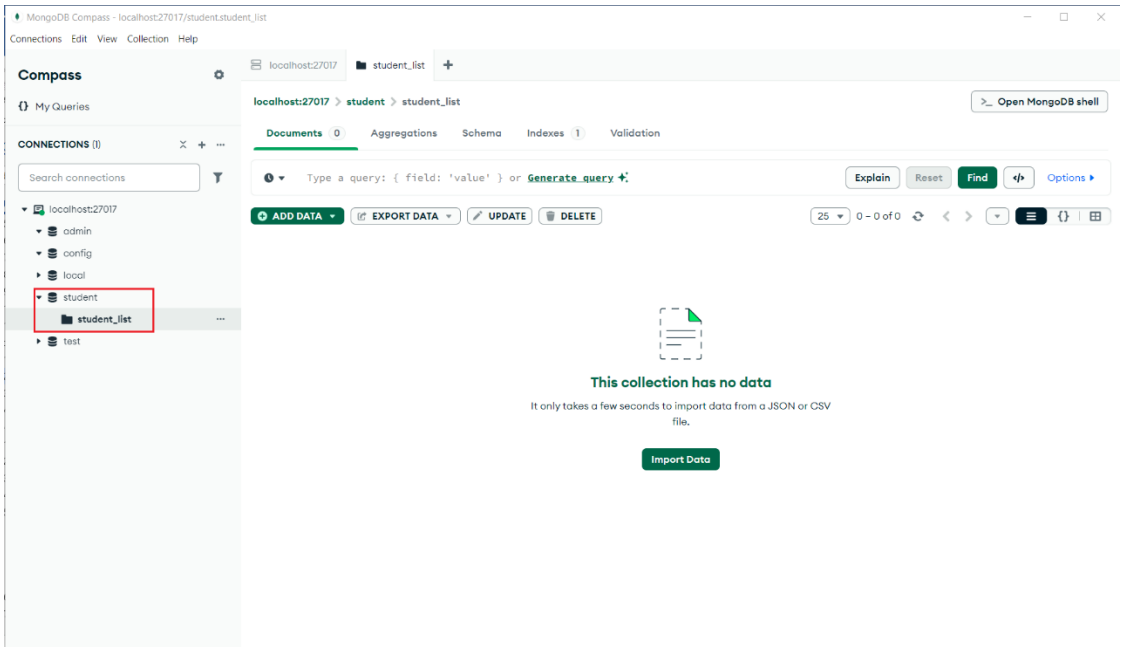
To create a new database, click on **+** icon next to `localhost:27010` connection.



Specify the database name (such as `student`) and collection name (such as `student_list`) and press **Create Database**.



You can see that the new database and collection has been created.



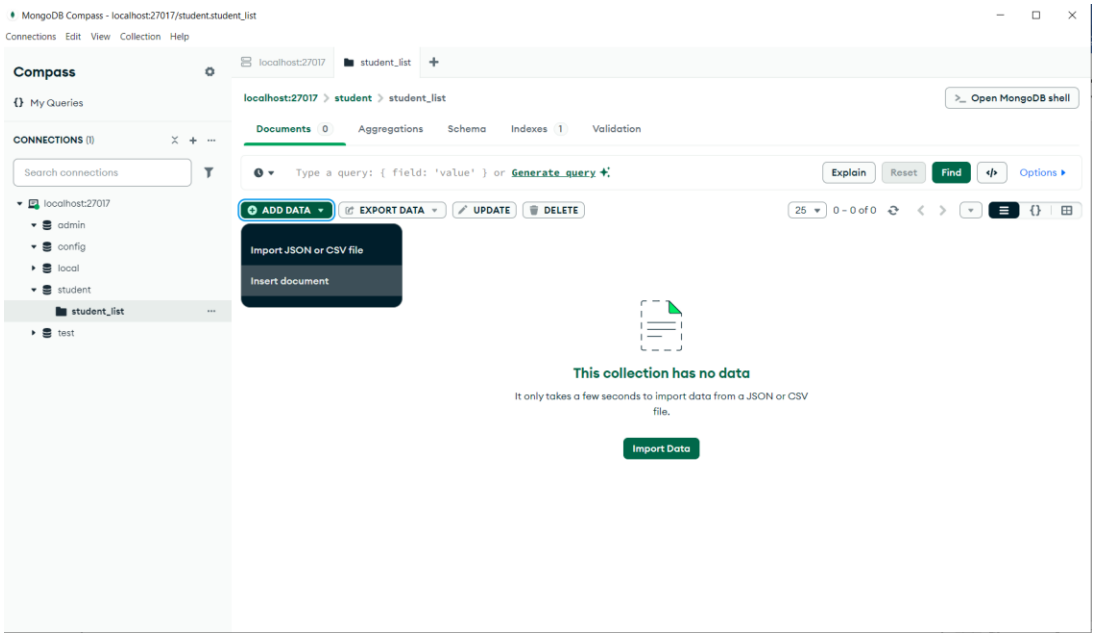
8.3. Insert Documents:

MongoDB Compass provides two ways to insert documents into a collection:

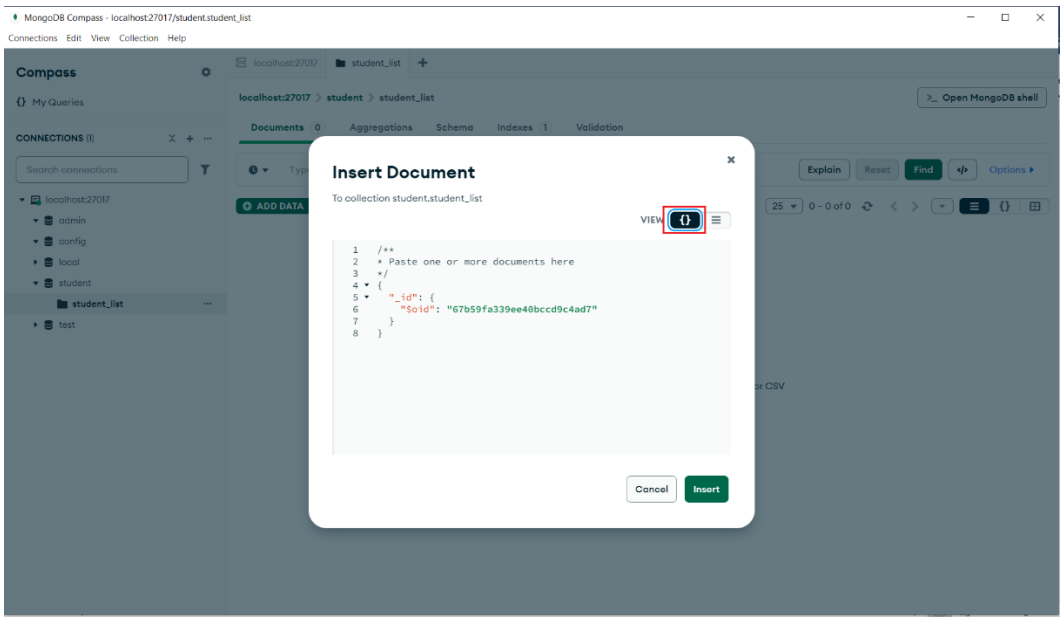
1. **Field-by-Field Editor** – You can use this option to insert a single document by selecting individual field values and types.
2. **JSON Mode** - You can use this option to insert multiple documents at once as an array. It allows you to write JSON documents in the editor.

Follow the below steps to insert documents into the collection:

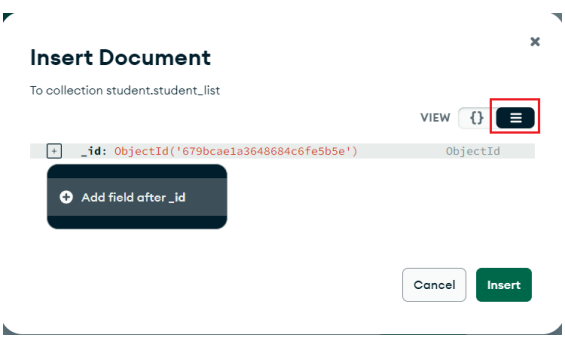
- Select `student_list` collection and click on **Add Data** dropdown and select **Insert Document** option.

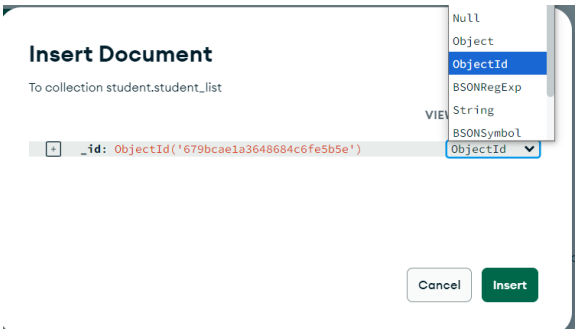


- Select the appropriate view based on how you would like to insert documents.
  Click the **{ }** brackets for JSON view which is the default view.
  Click the list icon for Field-by-Field editor.



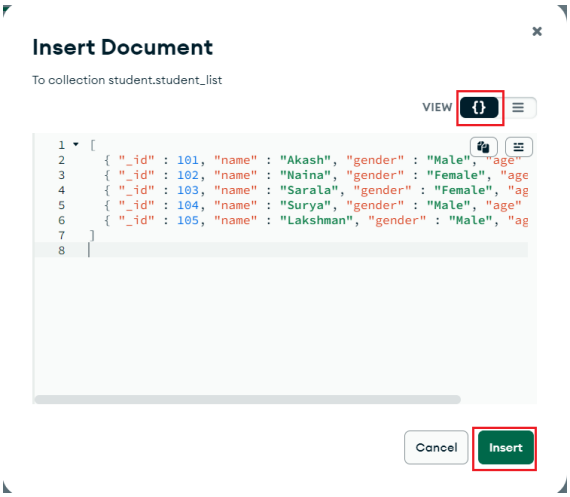
In the **Field-by-Field Editor**, ObjectID (`_id`) is generated by default and cannot be changed. Click on **+** icon before **_id** and select **Add field after _id** and enter the field name and value and choose the datatype of each field by clicking on drop down.



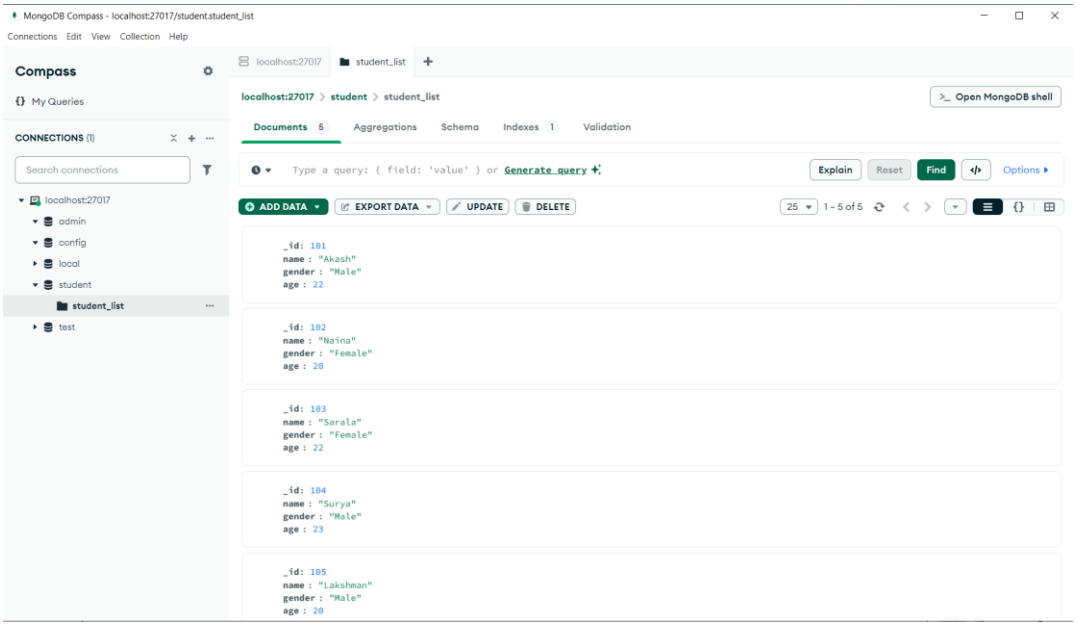


- In JSON view, enter the below documents to insert into the collection. Note that if you do not provide an ObjectId (`_id` field) in the document, Compass automatically generates an ObjectId for the inserted document.

```
[
  { "_id" : 101, "name" : "Akash", "gender" : "Male", "age" : 22
},
  { "_id" : 102, "name" : "Naina", "gender" : "Female", "age" :
20 },
  { "_id" : 103, "name" : "Sarala", "gender" : "Female", "age" :
22 },
  { "_id" : 104, "name" : "Surya", "gender" : "Male", "age" : 23
},
  { "_id" : 105, "name" : "Lakshman", "gender" : "Male", "age" :
20 }
]
```



- Once inserted, you can see the list of documents available in the collection



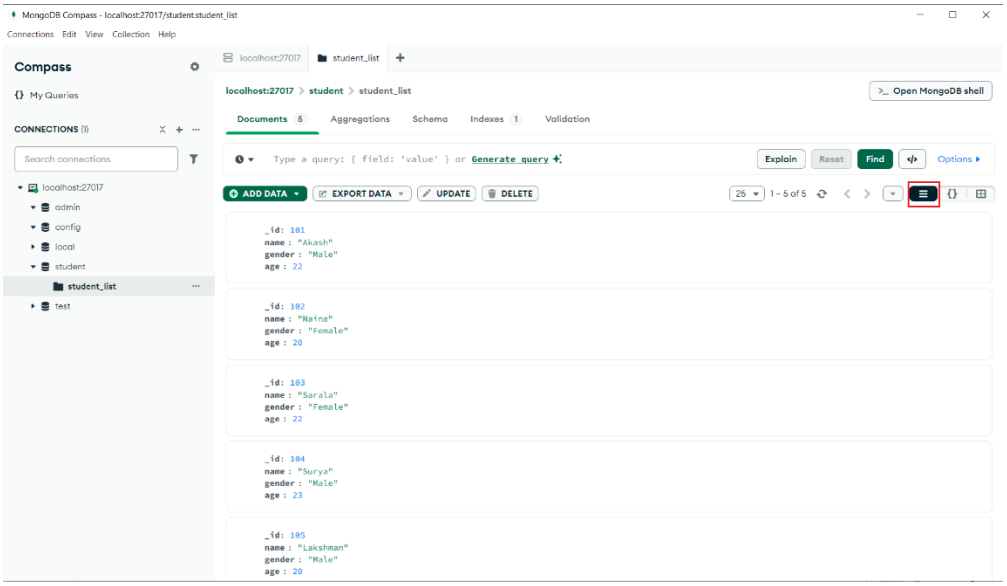
8.4. Modify Documents:

MongoDB Compass provides three ways to modify documents in a collection:

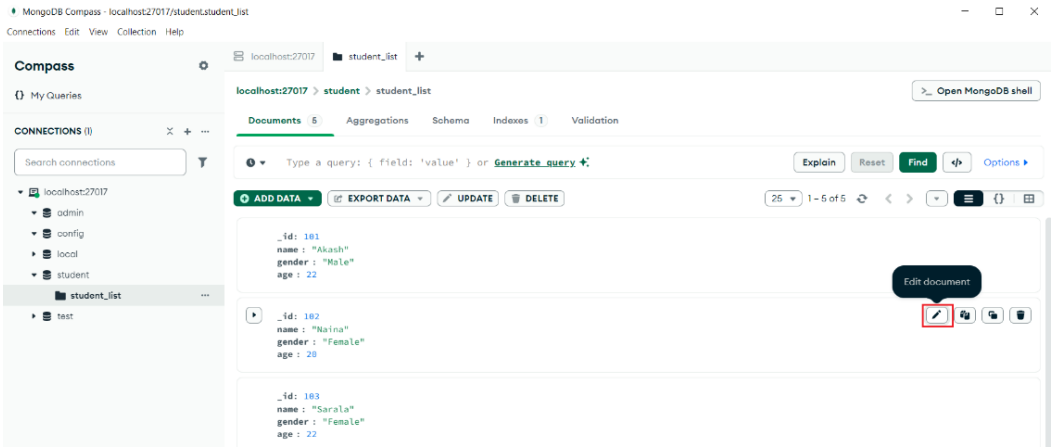
1. **List view** – This is the default view where you can view document content in the form of a list. When you edit a document in this view, Compass performs a `findOneAndUpdate` operation and updates only those fields that you have changed.
2. **Table view** – This view shows you the document content in a table format. When you edit a document in this view, Compass performs a `findOneAndUpdate` operation and updates only those fields that you have changed.
3. **JSON Mode** – This view shows you the document content in JSON format. When you edit a document in this view, Compass performs a `findOneAndReplace` operation and replaces the entire document.

Follow the below steps to modify single document in the collection:

- Select the appropriate tab based on whether you are viewing your documents in List, JSON, or Table view. Here, let's choose the default **List view**:

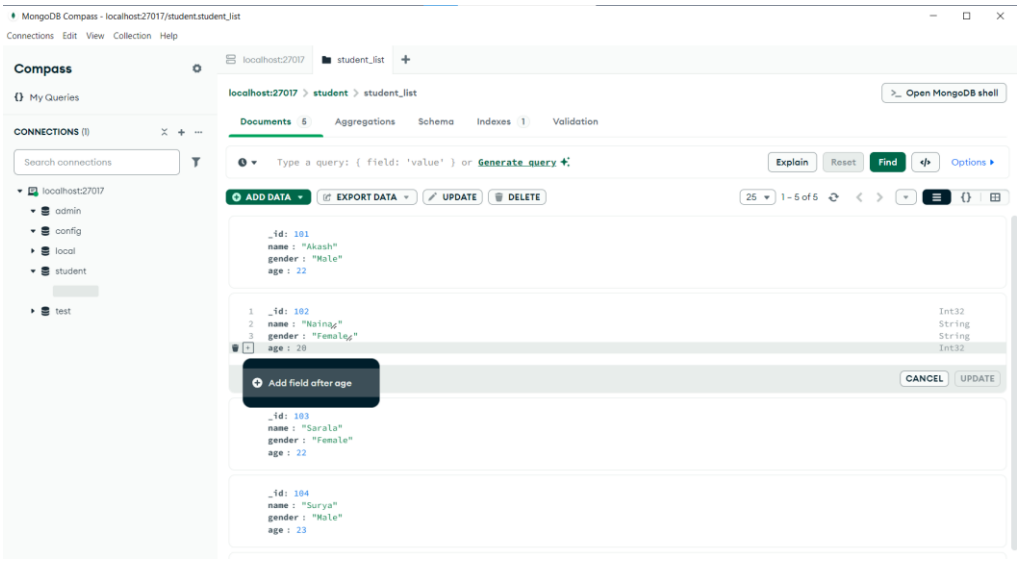


- To modify a document, hover over the document and click the pencil icon.

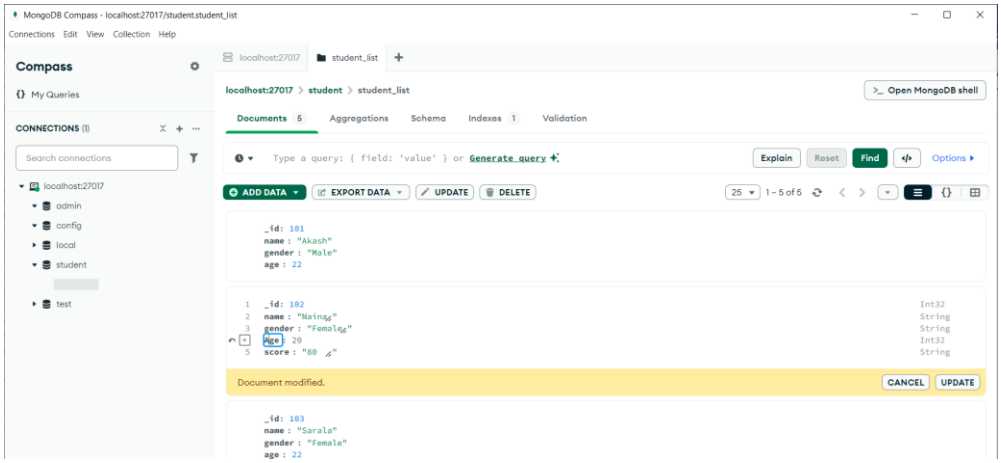


- After clicking the pencil icon, the document enters into the edit mode where we can make changes to the fields, values, or data types of values.

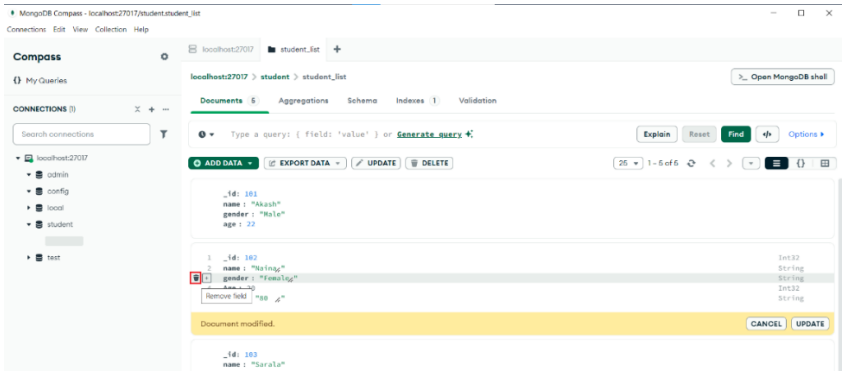
  To add a new field in the document after an existing field, hover over the field and click on the plus sign.
  For example, select the document with `_id:102` in `student_list` collection and click on **+** sign to the left of `age` field and you will see an option **Add field after age**.
  Let's, enter the new field name as `score` and value as `80`.



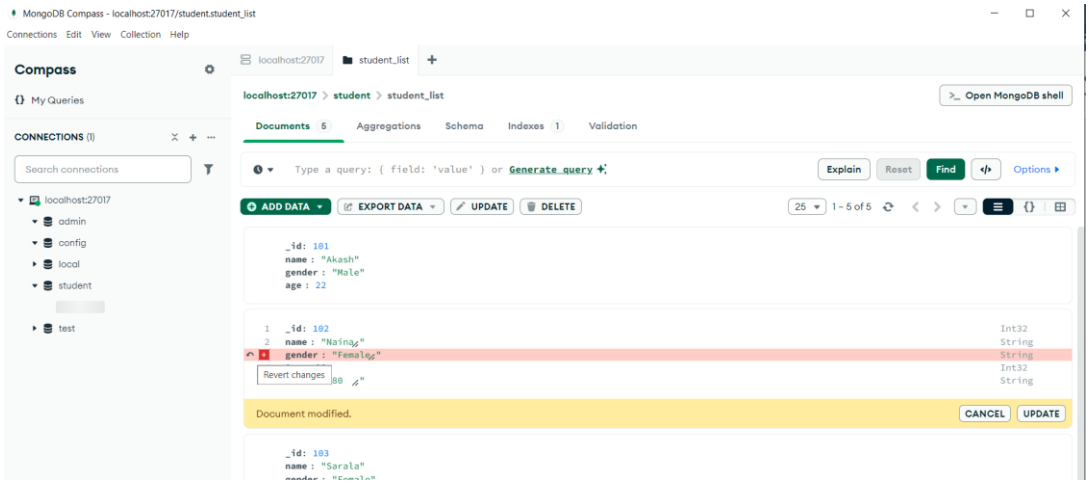
To modify an existing field name or value in the document, click on the respective field name or value.
For example, click on `age` field and change its name to `Age`



To delete a field from a document, click the delete icon to the left of the field:

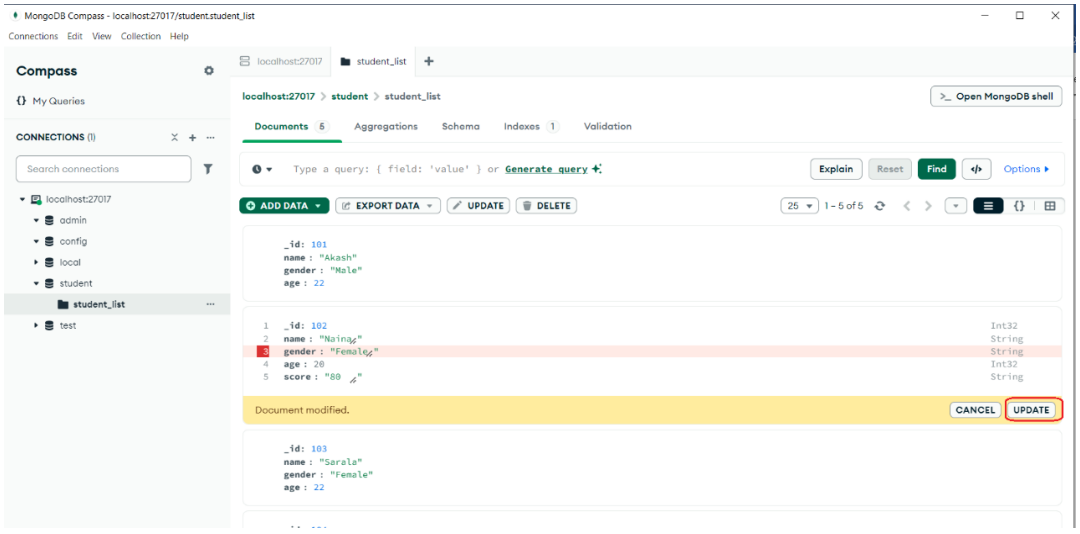


Once deleted, the field is marked for removal and appears highlighted in red. To revert changes to a document, hover over the edited field and click the revert icon which appears to the left of the field's line number.

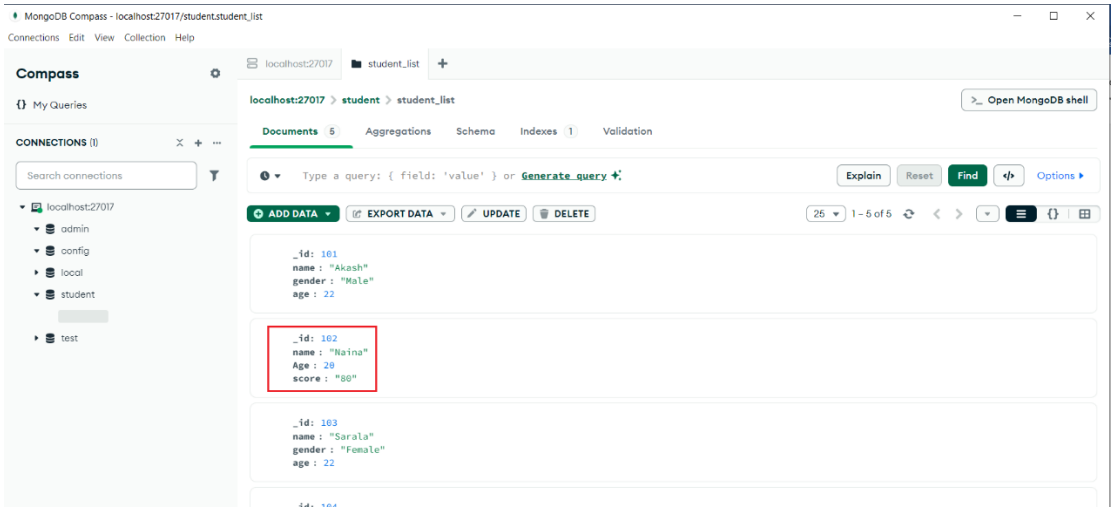


If you want to exit the edit mode and cancel all pending changes to the document, you can click the **Cancel** button.

When you are finished editing the document, click the **Update** button to commit your changes.



- Once changes are updated, you can see it in the collection.

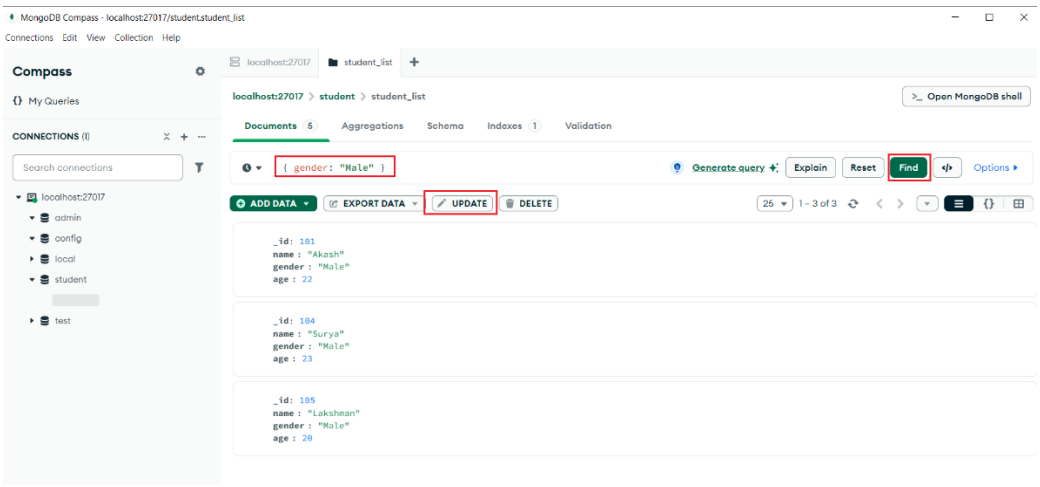


Follow the below steps to modify multiple documents in the collection. We can perform bulk update operations on multiple documents by using **Update Documents** modal.

- From the **Documents** tab, enter the filter criteria on the Query bar. The filter criteria specified applies to the documents in the **Bulk Update** modal. If you want to update all documents in a collection, leave the **Query bar** blank.

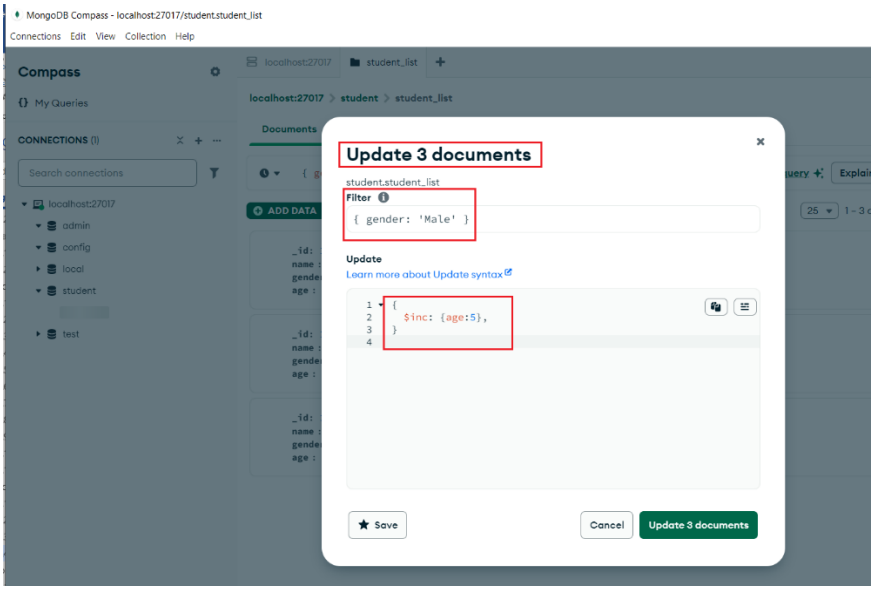
  For example, enter the following filter in the **Query bar** to filter male students and press **Find** button. Then press the **UPDATE** button to display the **Update Documents** modal.

```
{ gender: "Male" }
```



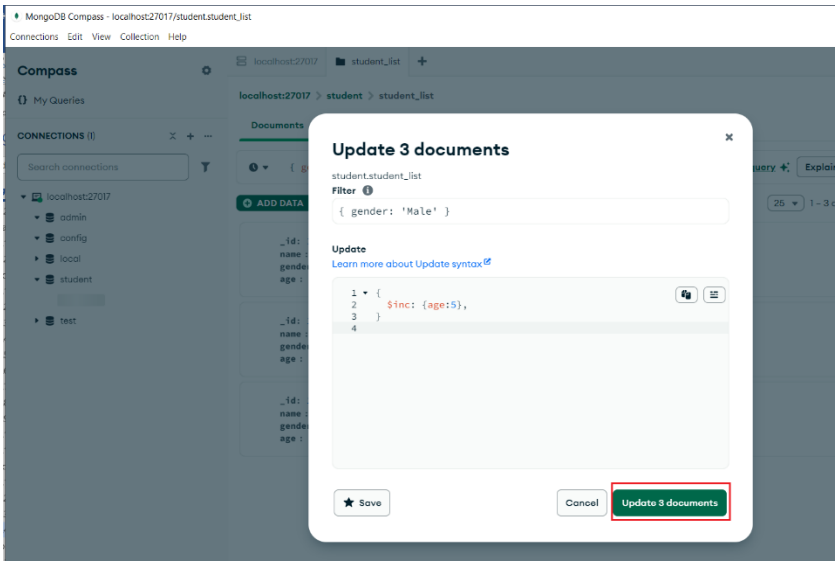
- The **Update Documents** modal displays the number of documents affected by the update at the top. In the modal, you can see the **Filter** field which displays the filter criteria specified on the Query bar. In the **Update** text field, enter the following syntax to increment the `age` by `5`.

```
{
  $inc: {age:5},
}
```

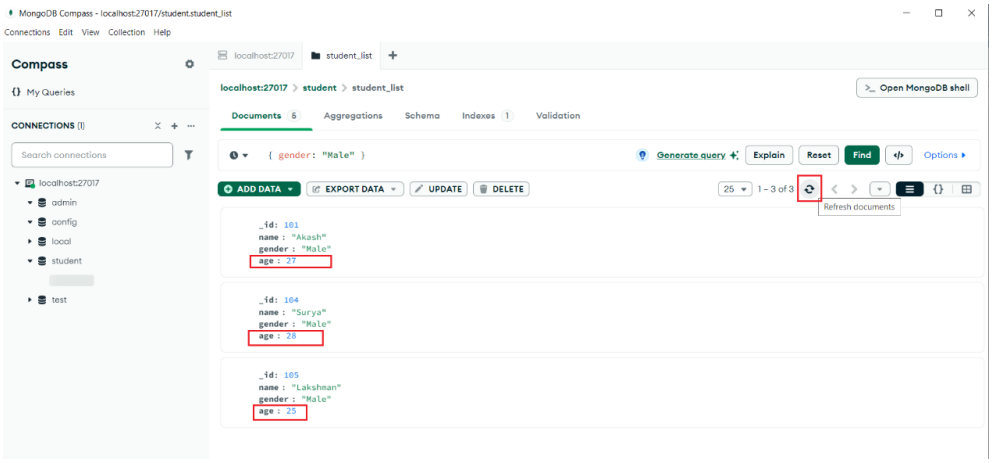


Optionally, you can save the update query in the **Update Documents** modal. Saving the query adds it to your favorite queries for that collection and allows you to load and copy the query after you close the modal. When you click the **Save** button on the bottom left of the modal, it asks you to enter a name for the update syntax and click the **Save** button.

Click **Update Documents** button to update the documents.



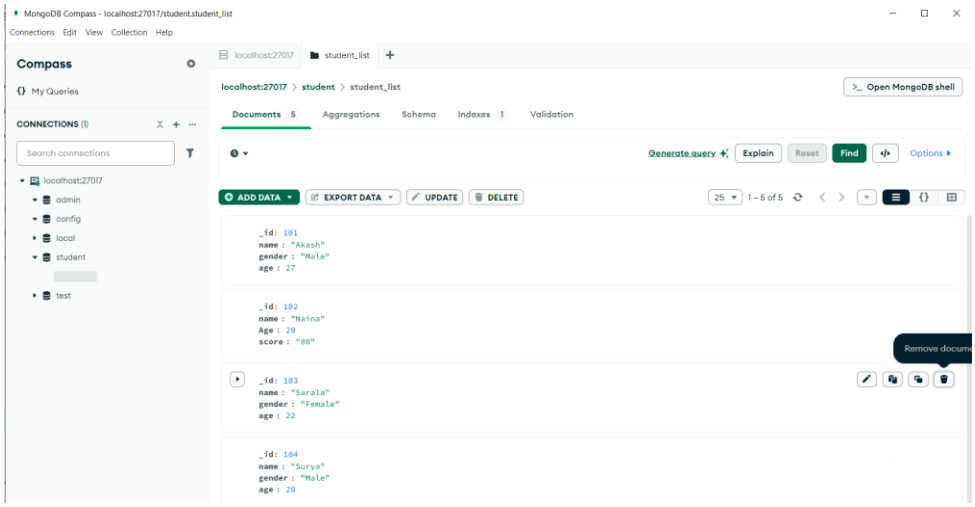
- On the **Documents** tab, click on **Refresh** icon to see the modified age of 3 male students.



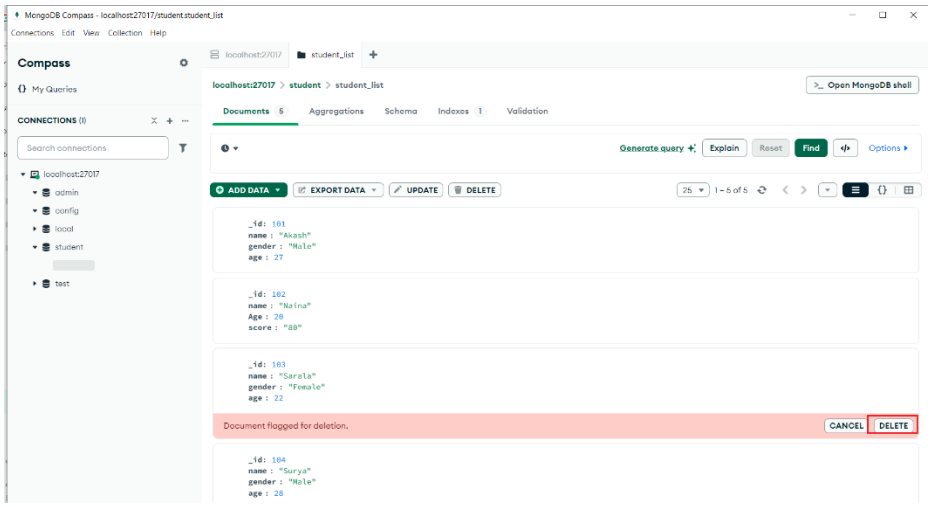
8.5. Delete Documents:

Follow the below steps to delete a single document in the collection:

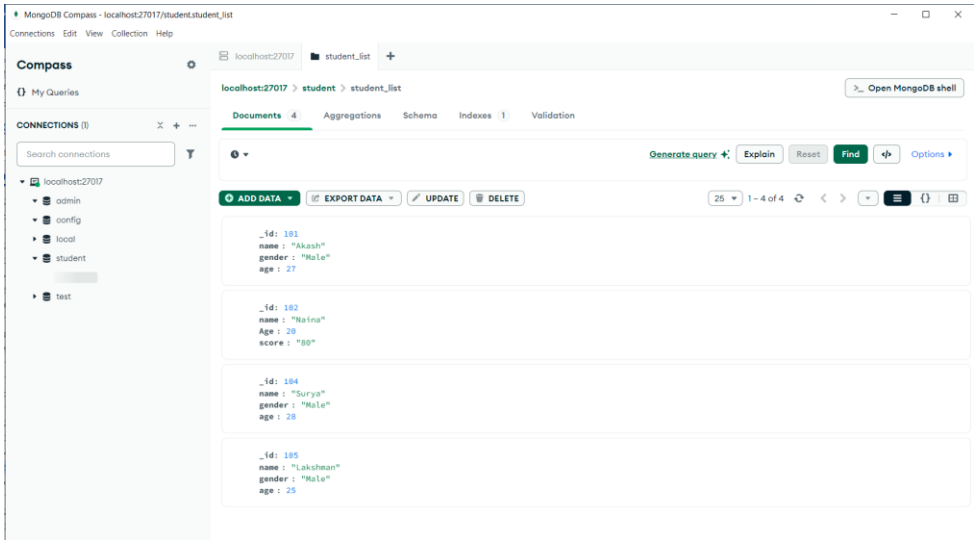
- To delete a document, hover over the document and click the delete icon.
  For example, select the document with `_id:103` in `student_list` collection and click on  .



- After clicking the delete icon, the document is flagged for deletion. Click on **Delete** to confirm the change.



- Once the document is deleted, you can see it in the collection.

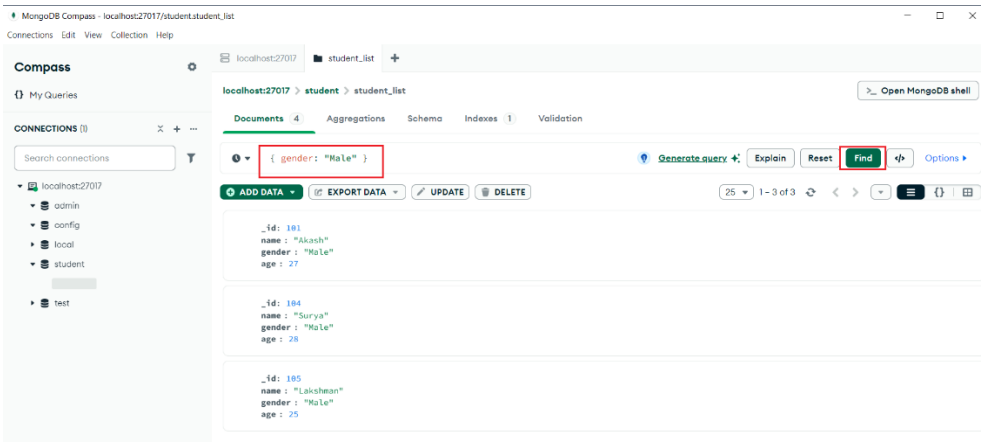


Follow the below steps to delete multiple documents in the collection. We can perform bulk delete operations by using **Delete Documents** modal.

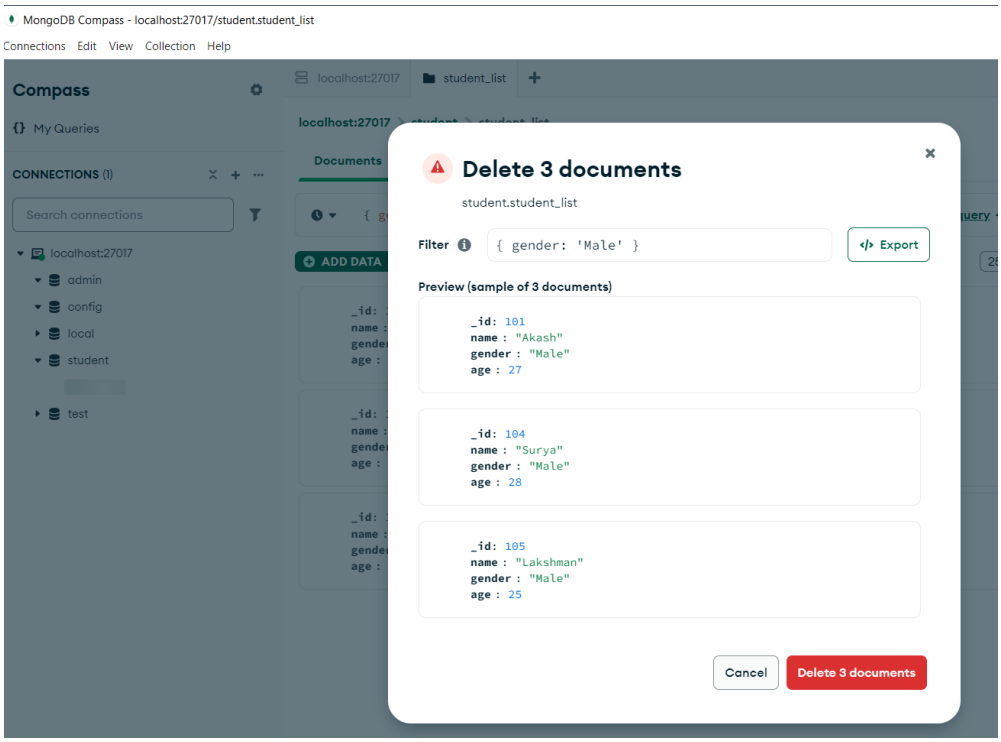
- From the **Documents** tab, enter the filter criteria on the Query bar to filter documents to be deleted. If you want to delete all documents in a collection, leave the **Query bar** blank.

  For example, enter the following filter in the **Query bar** to filter male students and press **Find** button. Then press the **DELETE** button to display the **Delete Documents** modal.

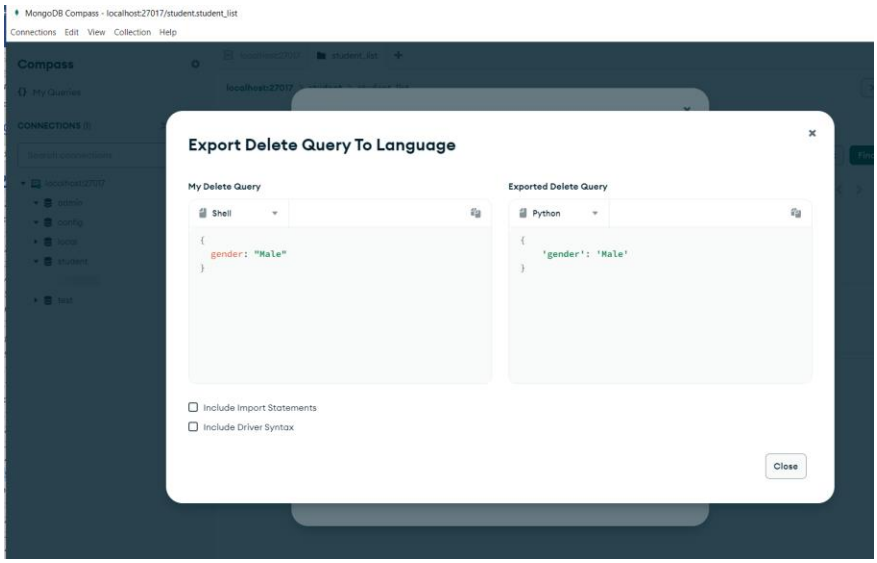
```
{ gender: "Male" }
```



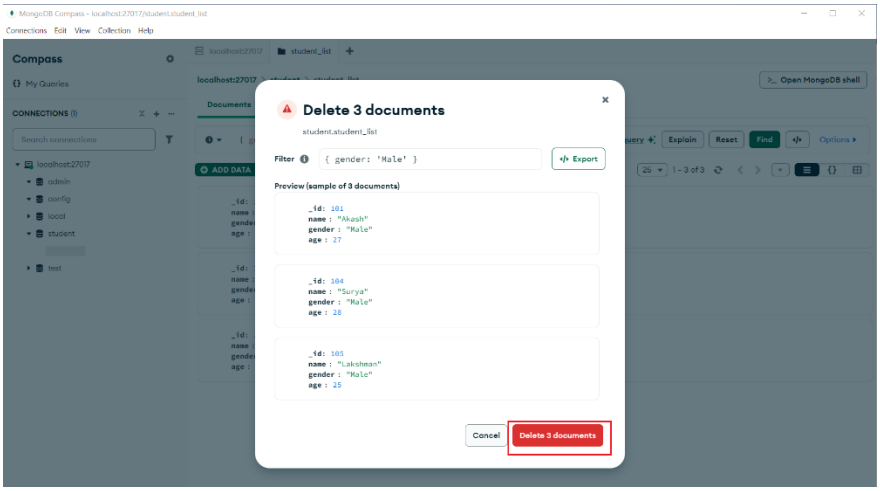
- The **Delete Documents** modal displays the number of documents affected by the delete at the top. In the modal, you can see the **Filter** field which displays the filter criteria specified on the Query bar and the **Preview** field with a preview of documents that will be deleted.



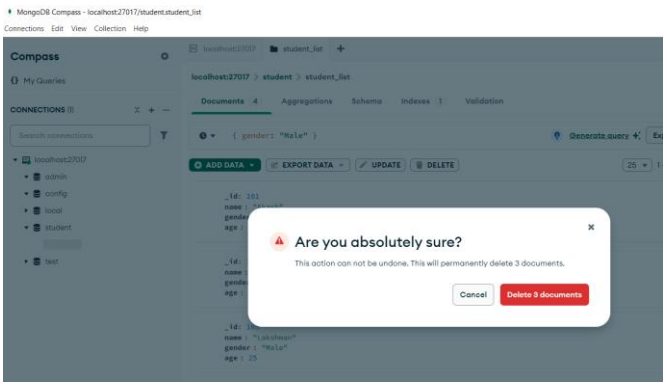
- Optionally, you can export the delete query to a supported driver language such as C#, Go, Java, Node, PHP, Python, Ruby, or Rust. On the **Delete Documents** modal, click **Export** which opens up the **Export Delete Query To Language** modal that displays with the delete syntax populated under **My Delete Query**. You can convert the delete syntax to the required programming language and copy the converted syntax.



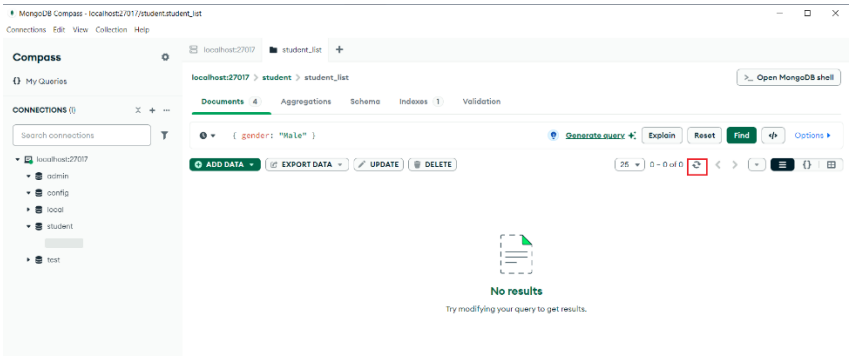
- On the **Delete Documents** modal, click **Delete Documents** button.



- Since the delete operation cannot be undone, it asks for your confirmation. Click on **Delete documents** to confirm the deletion.

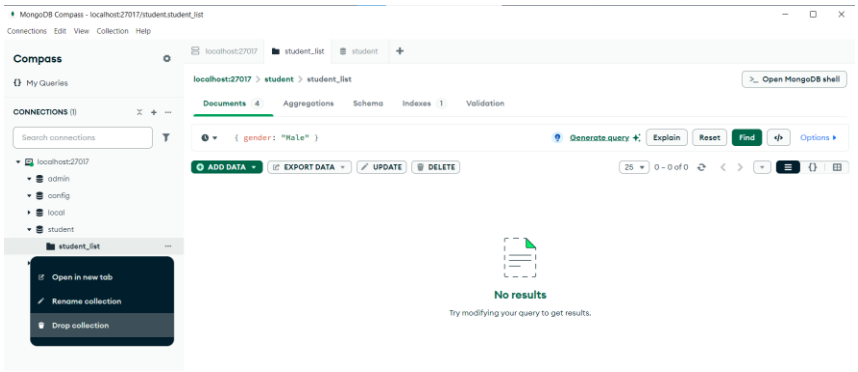


- On the **Documents** tab, click on **Refresh** icon and you can notice those documents were deleted.

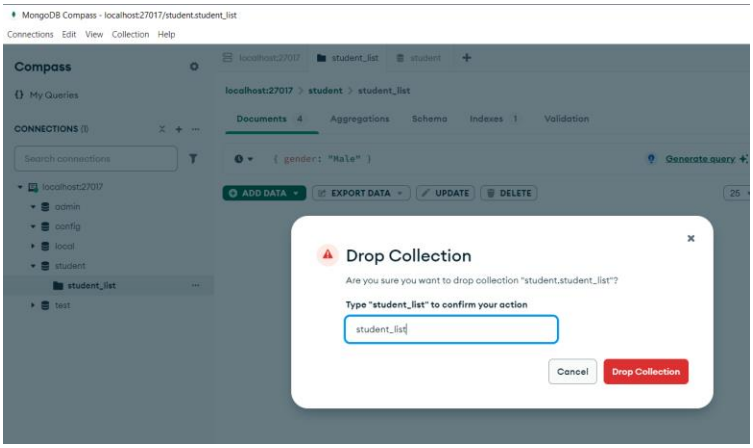


8.6. Drop Collection:

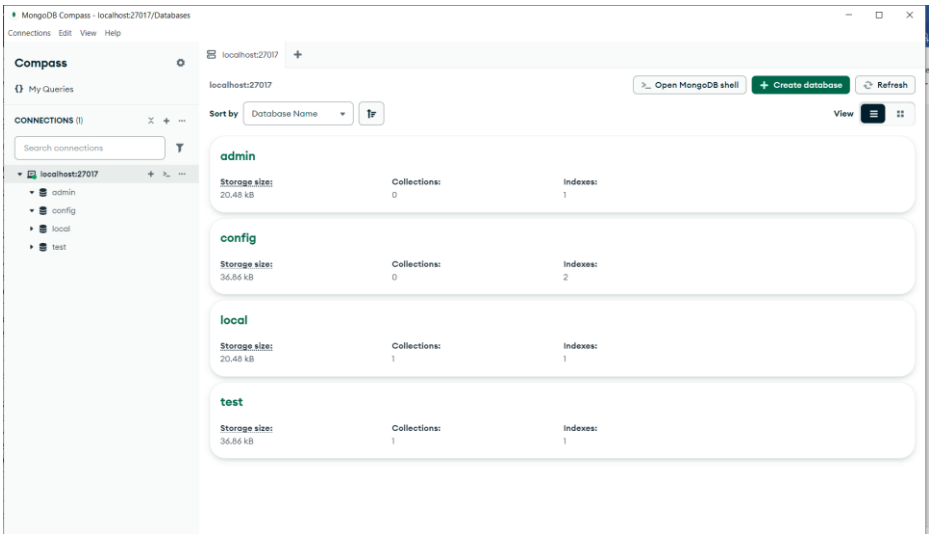
To drop a collection, click on 3 dots on the right of the `student_list` collection and choose **Drop collection**. Note that if the database consists of a single collection, the database will be auto-deleted when collection gets deleted.



Enter the collection name in the text field and click on **Drop Collection**.



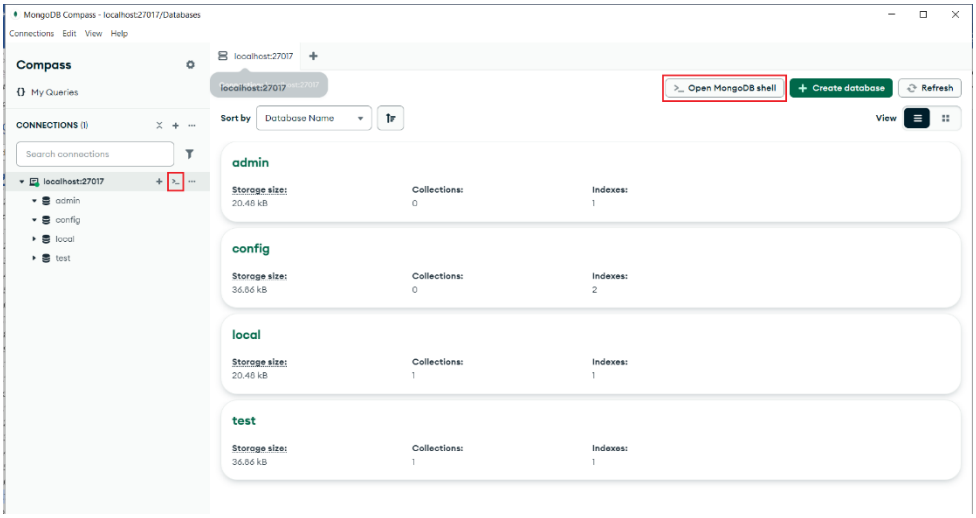
You can see that `student_list` collection and `student` database have been dropped.



8.7. Embedded MongoDB Shell:

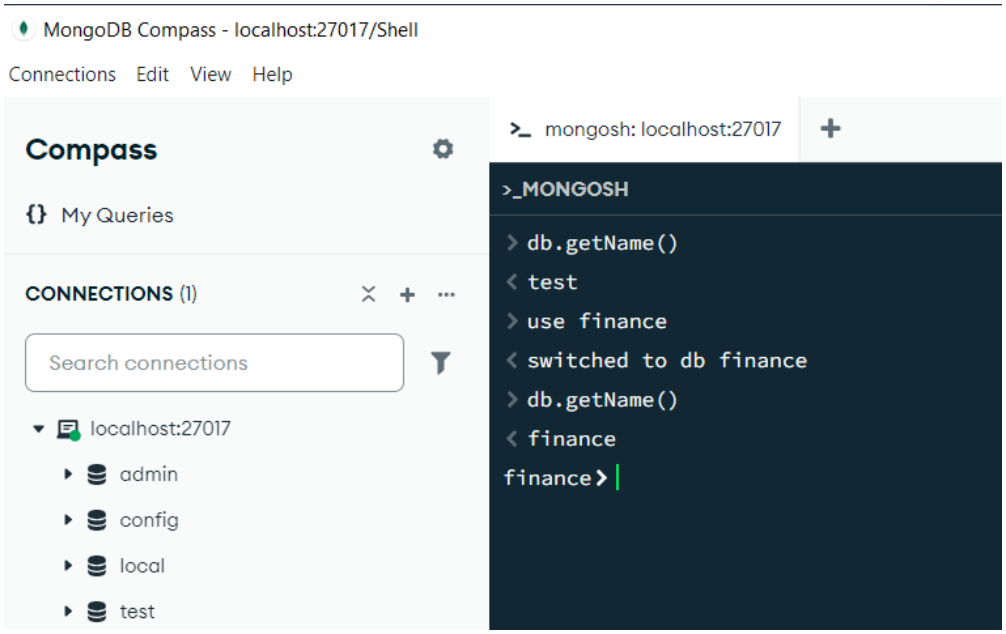
MongoDB Compass provides an embedded shell i.e mongosh, a JavaScript environment to interact with MongoDB instances. You can use mongosh to test queries and operations in your database.

To open the embedded mongosh, you can either click on **>_** icon on the right of your deployment name in the **Connections** sidebar or click on **>_Open MongoDB shell** in the top right of any tab connected to a MongoDB deployment.

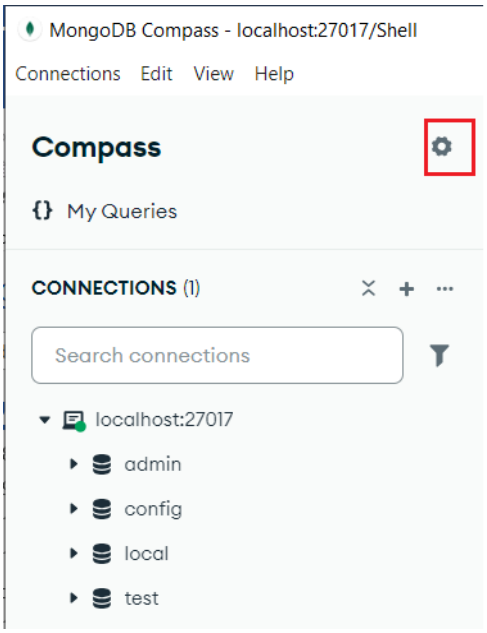


By default, mongosh connects to the `test` database. To use a different database, run the following command in mongosh and proceed with executing queries:

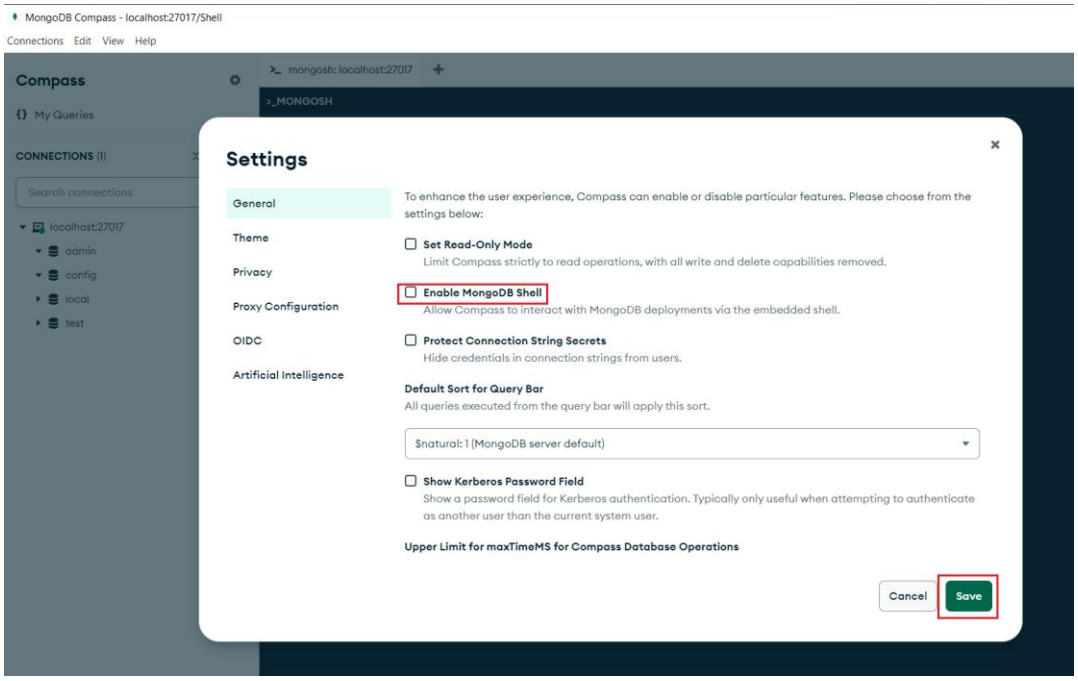
```
use database_name
```



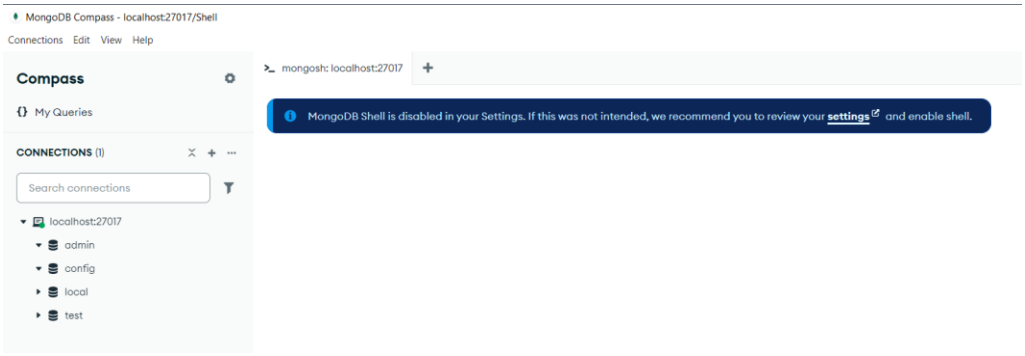
To disable the embedded MongoDB shell, click on **Settings** icon next to **Compass**.



On the **Settings** dialog box, go to **General** tab and uncheck **Enable MongoDB Shell** option and press **Save.**



Once you disable it, you will see an error "**MongoDB Shell is disabled in your Settings"** on the mongosh tab.



9. Deploy MongoDB Replica Set:

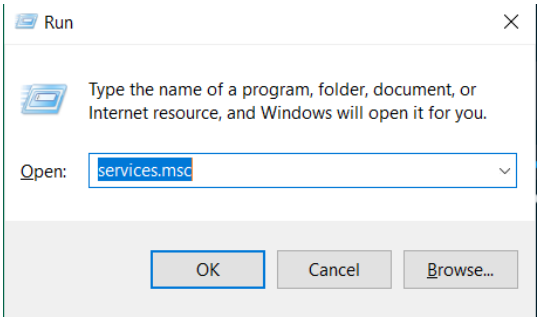
Now, we will see how to create a replica set with a primary, 2 secondary and one arbiter nodes.

For the purpose of this documentation, we will setup one arbiter and two secondary MongoDB instances running on different IP addresses and ports in the same Windows system but in the real-time project, we should configure one instance per system to setup Replica Set.

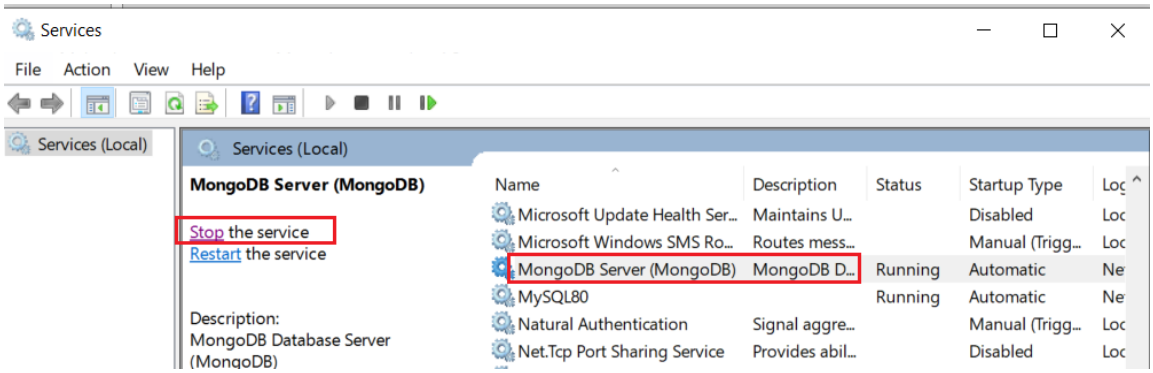
9.1. Shutdown Standalone Instance:

First, we should stop the standalone instance if it is already running.

Press **Windows +R** in your windows system to open **Run** application and type `services.msc` and press OK.

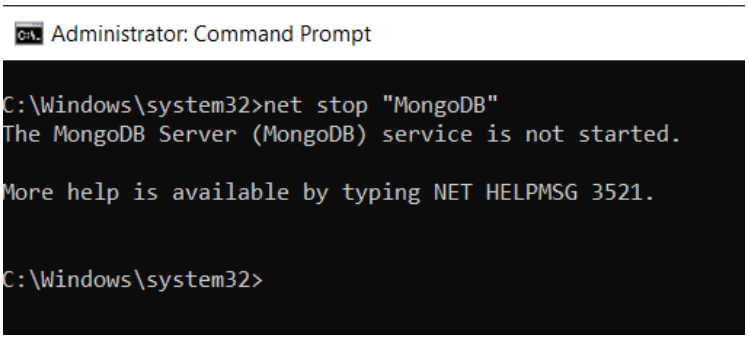


On the **Services** application, look for **MongoDB Server** service and review the **Status** which shows **Running**. Click on **Stop** link to stop the service.



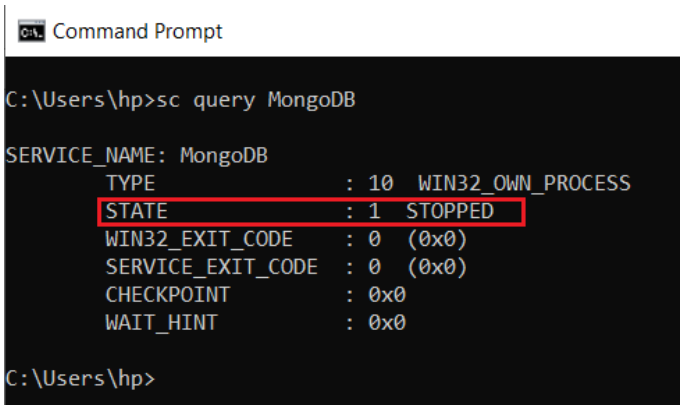
You can also stop the MongoDB service through **Command Prompt** application as **Administrator** using the below command:

```
net stop "MongoDB"
```



Verify if the service is stopped successfully using the below command:

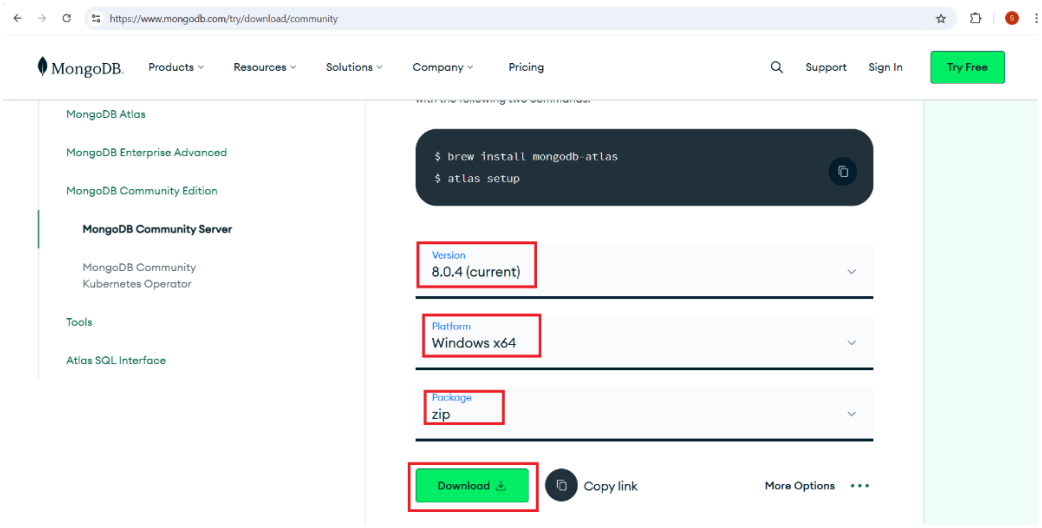
```
sc query "MongoDB"
```



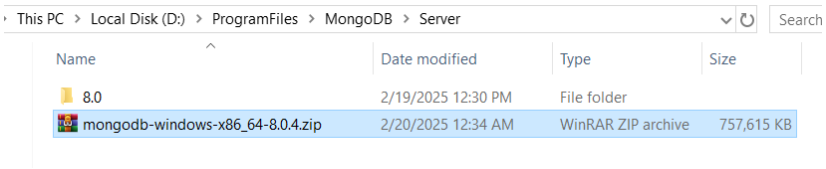
Here, you can see the STATE as **STOPPED**.

9.2. Setup Multiple Instances:

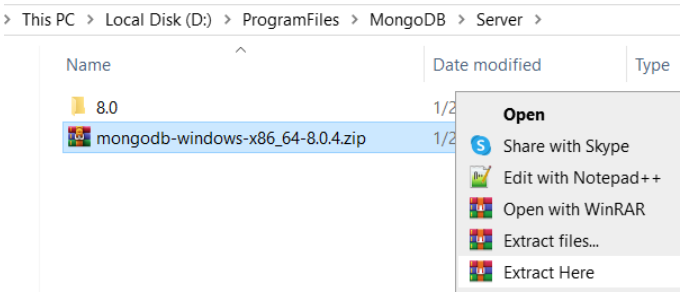
Go to [MongoDB Community Server Download Center] page. Choose the latest **Version** *(at the time of writing this document, the latest version is 8.0.4)* and **Platform** as `Windows X64` and **Package** as `zip` and click on **Download** button which downloads the file into your **Downloads** folder in your machine.



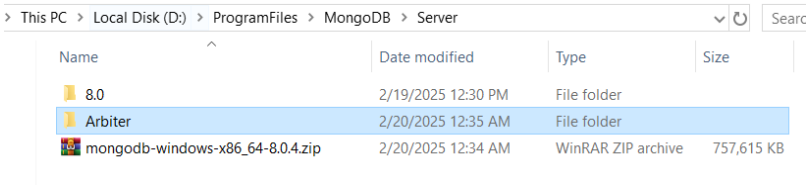
Copy the downloaded file `mongodb-windows-x86_64-8.0.4.zip` from your **Downloads** folder into MongoDB installation path at `D:\ProgramFiles\MongoDB\Server`.



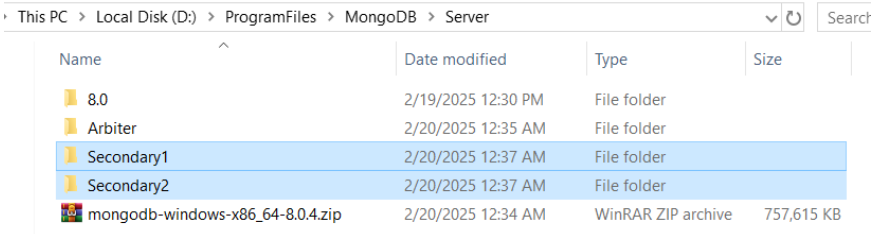
Right click on the file and select **Extract Here** option to extract contents of it into the current directory.



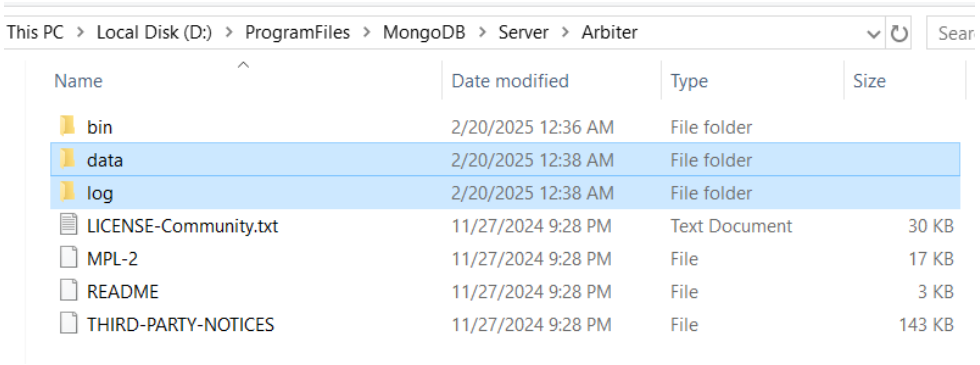
The zip file extraction may take a couple of minutes to finish. After finishing, you see a folder named `mongodb-win32-x86_64-windows-8.0.4` which consists of MongoDB binaries. Rename this newly created folder as `Arbiter`.

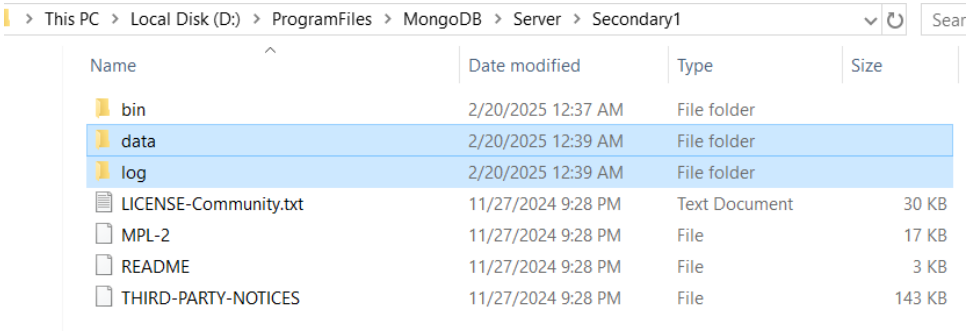


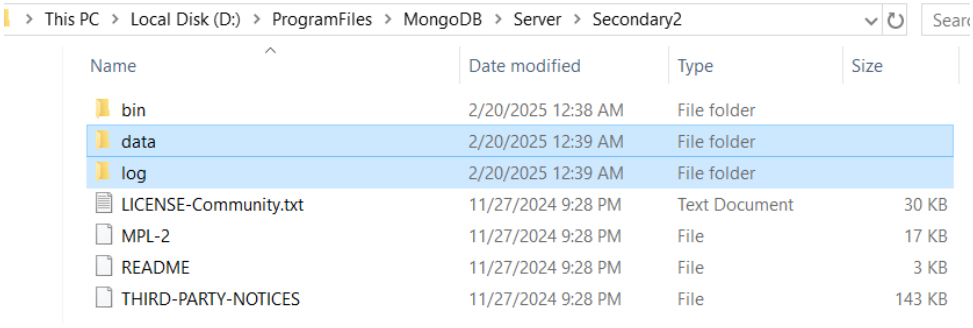
Take two copies of `Arbiter` folder and rename those copies as `Secondary1` and `Secondary2`.



Create two sub-directories named `data` and `log` under `Arbiter`, `Secondary1` and `Secondary2` directories. Note that arbiter uses `data` directory for configuration data and does not store the actual datasets but until the arbiter instance is added to the replica set, the arbiter acts like other instances and start up with a set of data files.





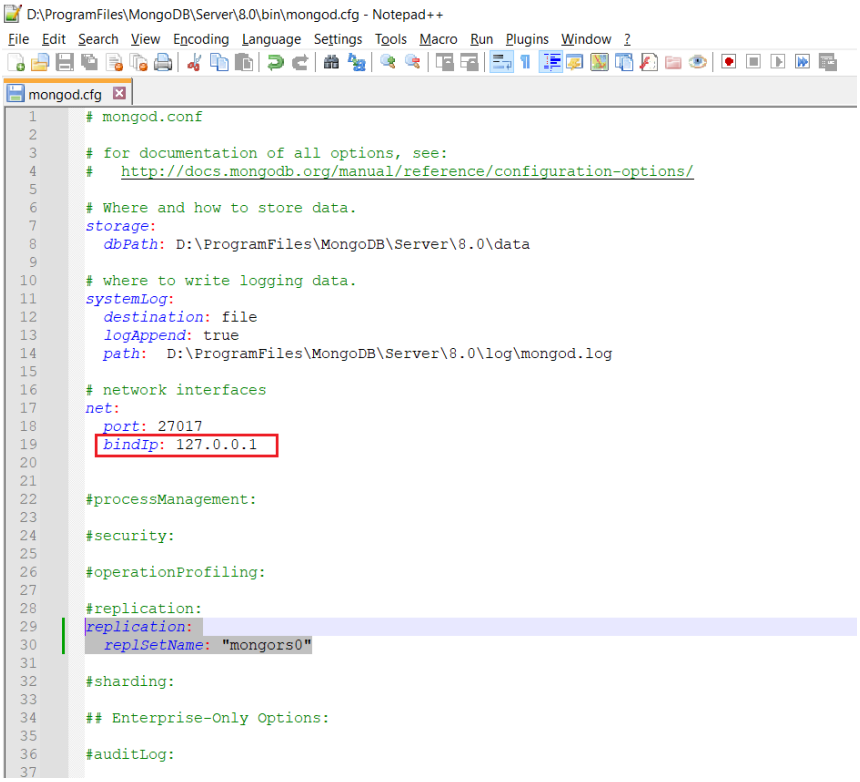


9.3. Set Replica Set Name:

Now, we are going to set our replica set name as `mongors0` in the configuration file for each MongoDB instance.

Open `mongod.cfg` file from the MongoDB installation location at `D:\ProgramFiles\MongoDB\Server\8.0\bin` and add the below lines to set the `replSetName` setting.

```
replication:
  replSetName: "mongors0"
```

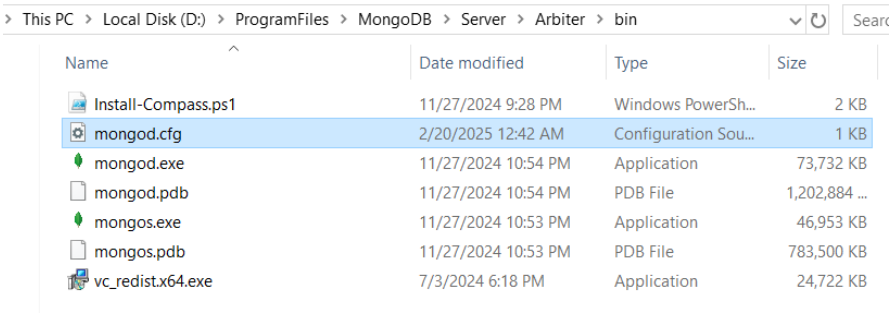


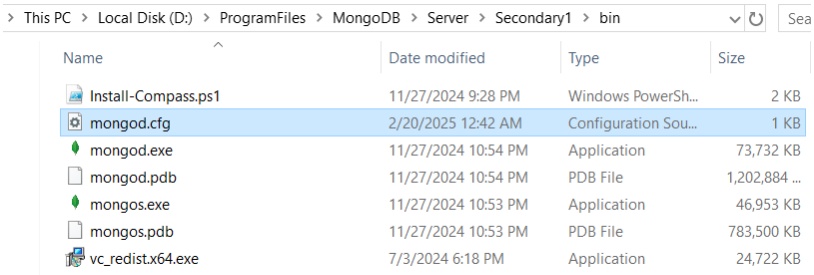
Note:

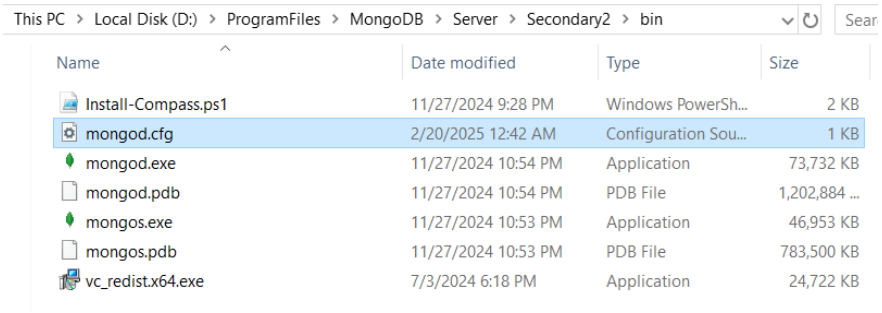
MongoDB binaries which are `mongod` and `mongos` are bound to `localhost` (IP address: `127.0.0.1`) by default, which is why you see `bindIP` is set to `127.0.0.1` in the above config file. When `mongod` and `mongos` are bound to `localhost` they can accept connections from clients running on the same machine which means remote clients cannot connect to binaries that are bound only to localhost. To override the default binding and bind to other IP addresses, update the `bindIp` setting in the configuration file with a list of hostnames or IP addresses. For now, this is not needed.

Next, copy the updated `mongod.cfg` file from `D:\ProgramFiles\MongoDB\Server\8.0\bin` and place it under `Arbiter`, `Secondary1` and `Secondary2` directories at the below locations

```
D:\ProgramFiles\MongoDB\Server\Arbiter\bin
D:\ProgramFiles\MongoDB\Server\Secondary1\bin
D:\ProgramFiles\MongoDB\Server\Secondary2\bin
```







9.4. Configure Replica Set Members:

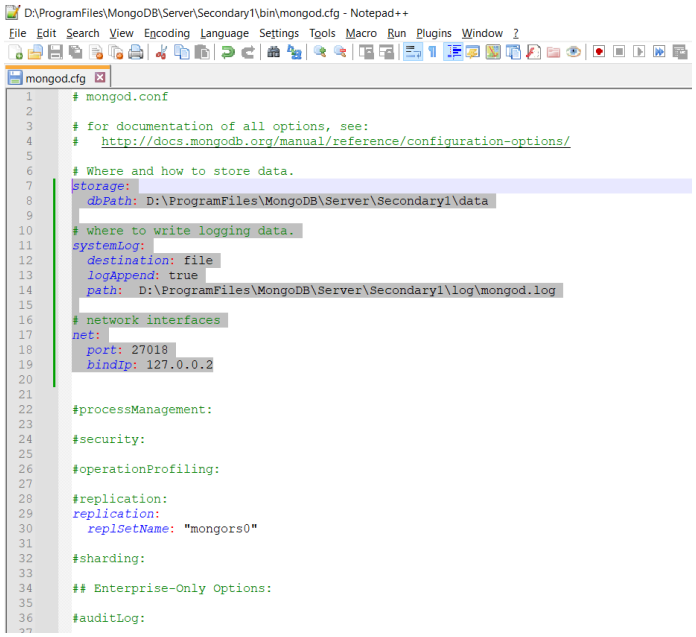
Let us configure our replica set members such that `Secondary1` instance runs at IP address `127.0.0.2` on port `27018`, `Secondary2` instance runs at IP address `127.0.0.3` on port `27019` and `Arbiter` instance runs at IP address `127.0.0.4` on port `27020` and also update the storage path and log locations of Secondary and Arbiter instances.

Note: Ensure that no process is running at `127.0.0.2:27018`, `127.0.0.3:27019` and `127.0.0.4:27020` ports in your Windows system.

Open `Secondary1` instance `mongod.cfg` from `D:\ProgramFiles\MongoDB\Server\Secondary1\bin` and update the following attributes:

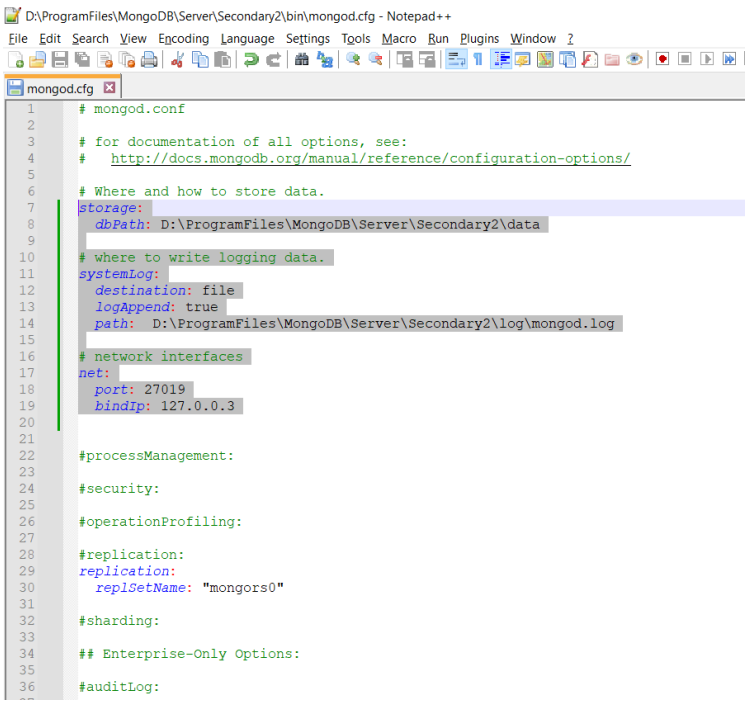
```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Secondary1\data

# where to write logging data.
```

```
systemLog:
  destination: file
  logAppend: true
  path:  D:\ProgramFiles\MongoDB\Server\Secondary1\log\mongod.log

# network interfaces
net:
  port: 27018
  bindIp: 127.0.0.2
```



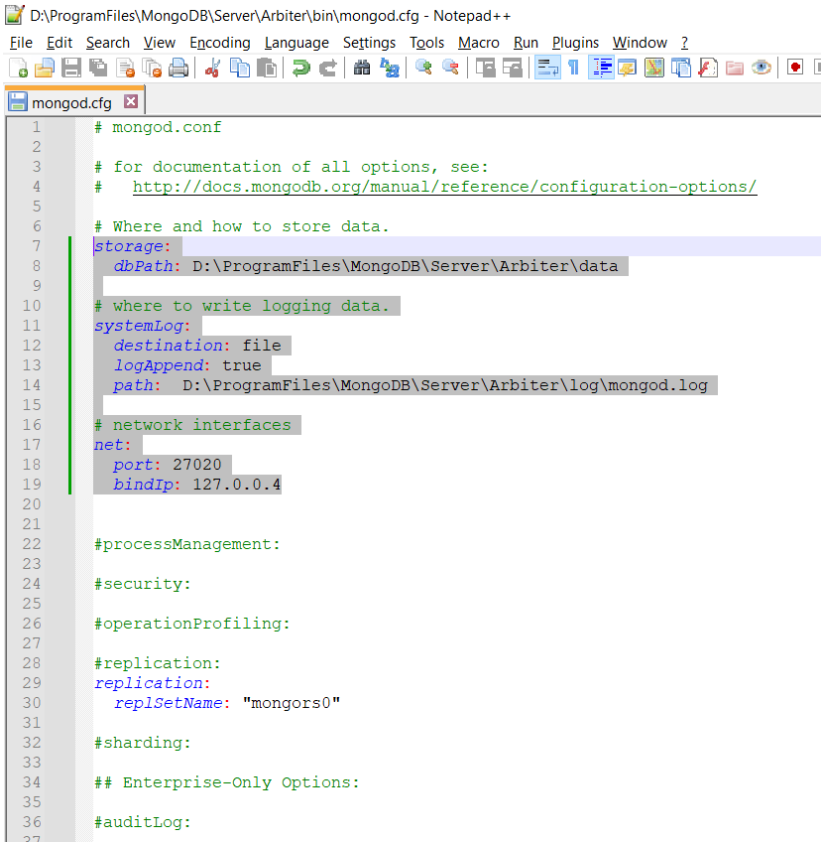
Similarly, open `Secondary2` instance `mongod.cfg` from `D:\ProgramFiles\MongoDB\Server\Secondary2\bin` and update the following attributes:

```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Secondary2\data

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path:  D:\ProgramFiles\MongoDB\Server\Secondary2\log\mongod.log

# network interfaces
net:
  port: 27019
  bindIp: 127.0.0.3
```



Similarly, open `Arbiter` instance `mongod.cfg` from `D:\ProgramFiles\MongoDB\Server\Arbiter\bin` and update the following attributes:

```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Arbiter\data

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path:  D:\ProgramFiles\MongoDB\Server\Arbiter\log\mongod.log

# network interfaces
net:
  port: 27020
  bindIp: 127.0.0.4
```



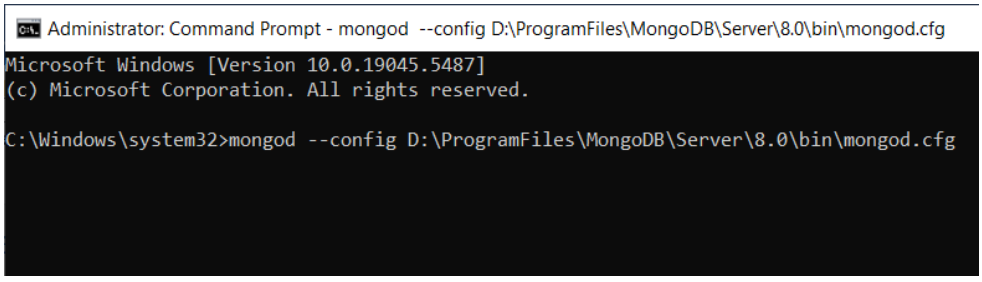
9.5. Start all MongoDB Instances:

Now, it's time to start all mongo DB instances including Primary, Secondary1, Secondary2 and Arbiter.

9.5.1. Start Primary Instance:

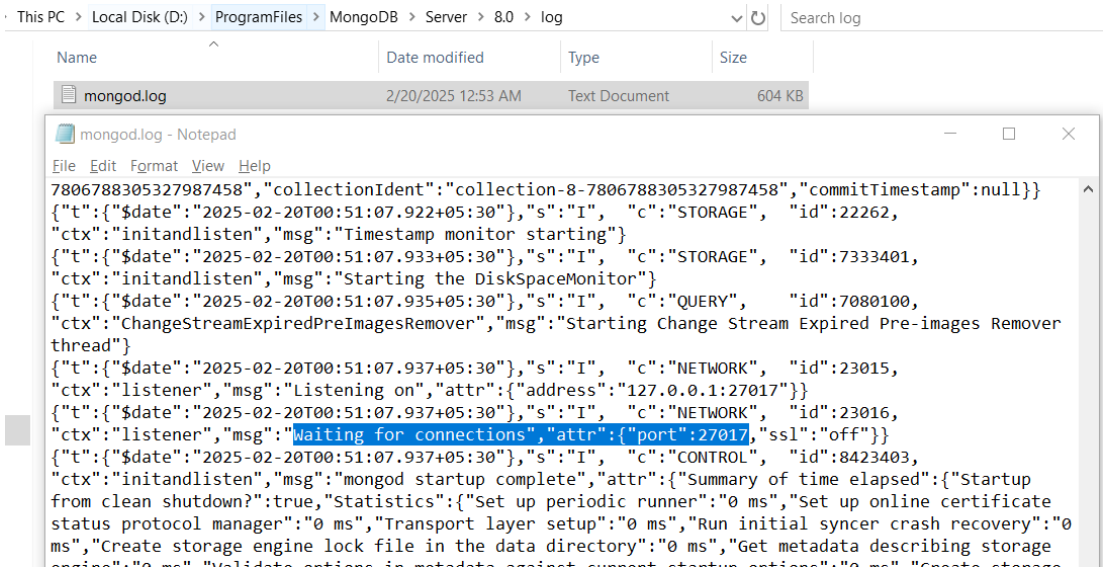
Open a new **Command Prompt** application as **Administrator** and execute the below command to start the first and primary instance.

```
mongod --config D:\ProgramFiles\MongoDB\Server\8.0\bin\mongod.cfg
```



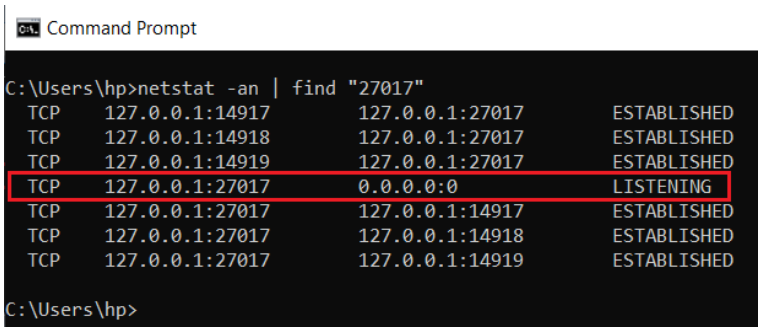
Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\8.0\log` and look

for messages **"Waiting for connections" port: 27017** and **"mongod startup complete"** which indicate that MongoDB instance has been started successfully.



You can also verify if the primary instance port `27017` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "27017"
```

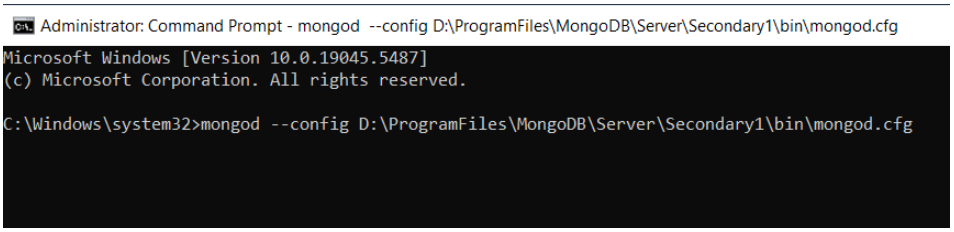


You can see that port `27017` is listening at IP address `127:0.0.1`.

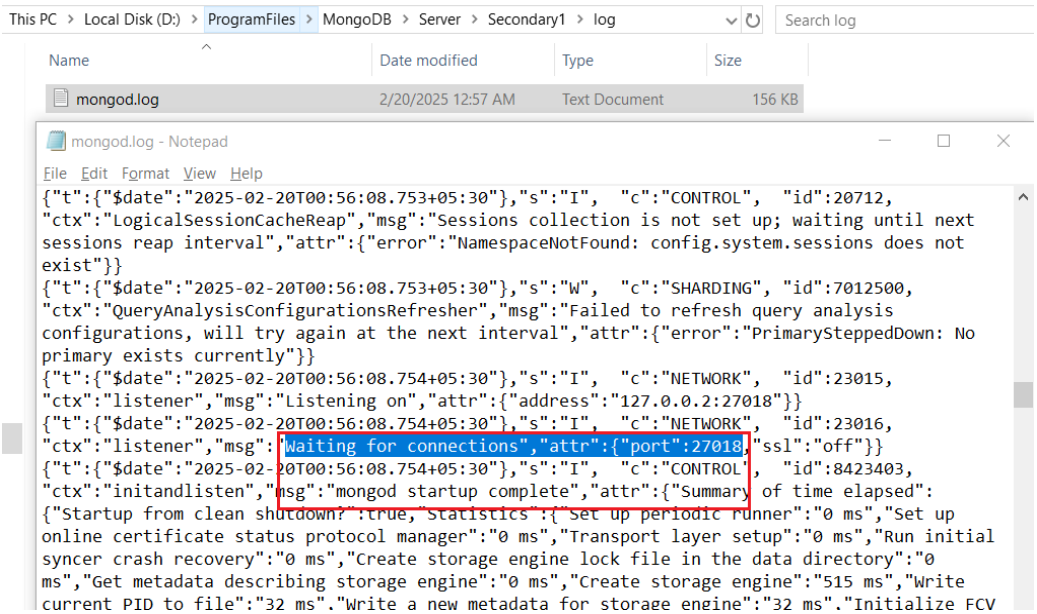
9.5.2. Start Secondary1 Instance:

Open a new **Command Prompt** application as **Administrator** and execute the below command to start the first secondary instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Secondary1\bin\mongod.cfg
```

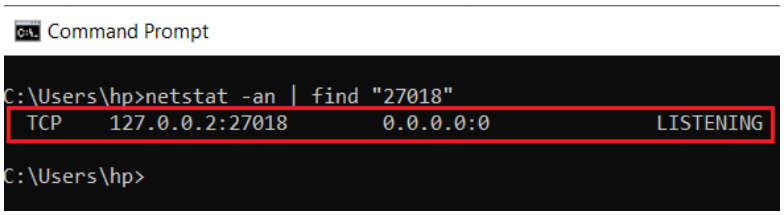


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Secondary1\log` and look for messages **"Waiting for connections" port: 27018** and **"mongod startup complete"** which indicate that MongoDB instance has been started successfully.



You can also verify if the secondary1 instance port `27018` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "27018"
```

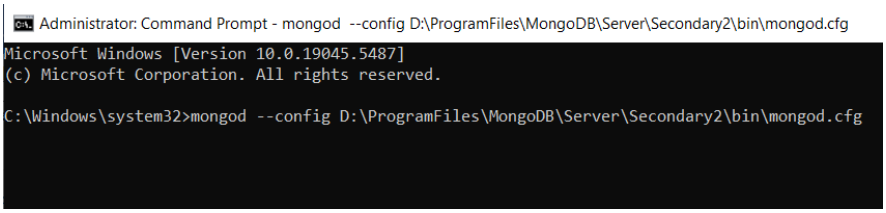


You can see that port `27018` is listening at IP address `127:0.0.2`.

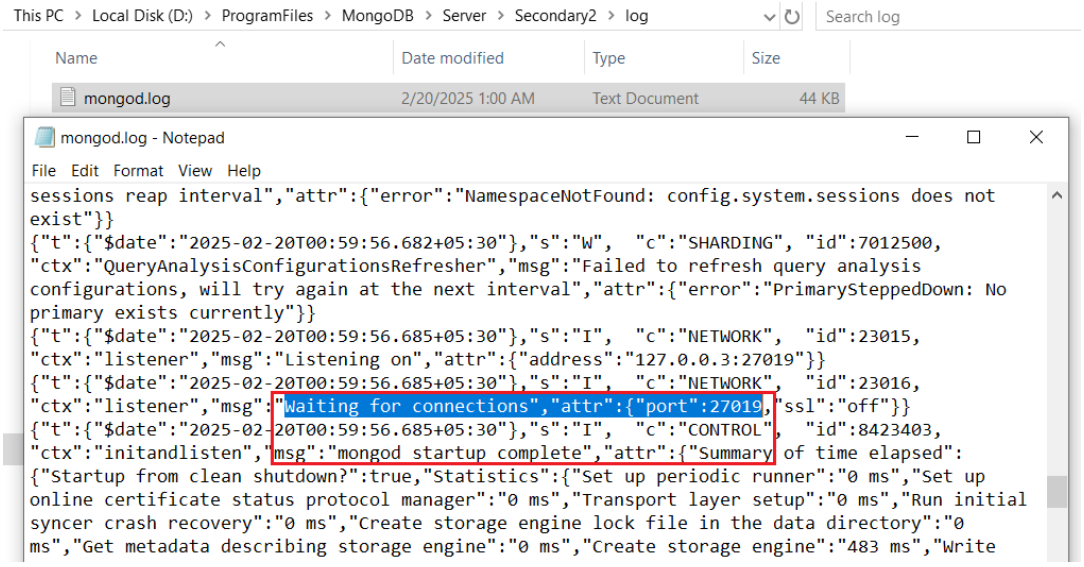
9.5.3. Start Secondary2 Instance:

Open a new **Command Prompt** application as **Administrator** and execute the below command to start the second secondary instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Secondary2\bin\mongod.cfg
```

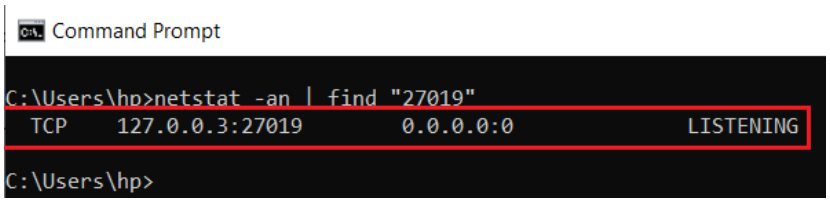


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Secondary2\log` and look for messages **"Waiting for connections" port: 27019** and **"mongod startup complete"** which indicate that MongoDB instance has been started successfully.



You can also verify if the secondar2 instance port `27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "27019"
```

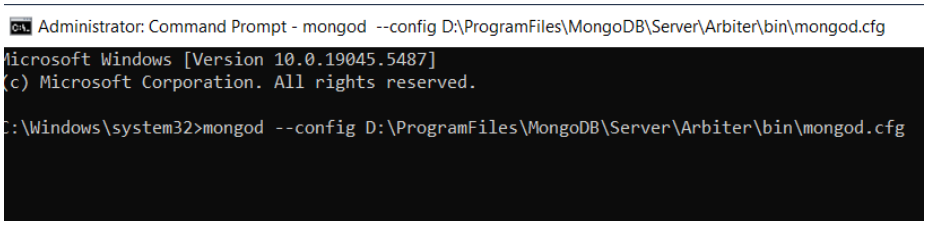


You can see that port `27019` is listening at IP address `127:0.0.3`.

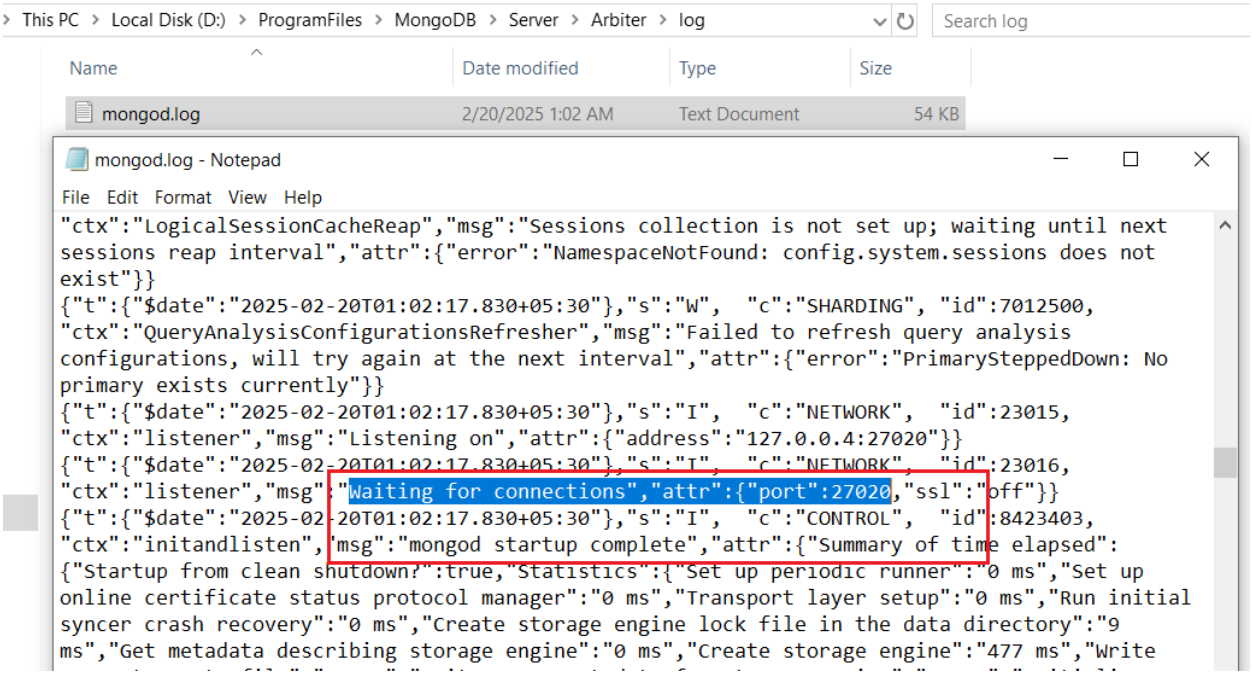
9.5.4. Start Arbiter Instance:

Open a new **Command Prompt** application as **Administrator** and execute the below command to start the arbiter instance.

```
mongod --config D:\ProgramFiles\MongoDB\Server\Arbiter\bin\mongod.cfg
```

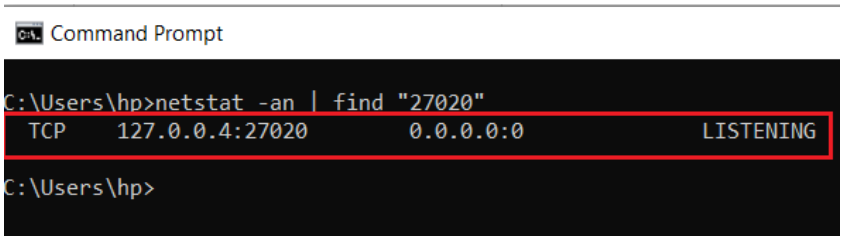


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Arbiter\log` and look for messages **"Waiting for connections" port: 27020** and **"mongod startup complete"** which indicate that MongoDB instance has been started successfully.



You can also verify if the arbiter instance port `27020` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "27020"
```

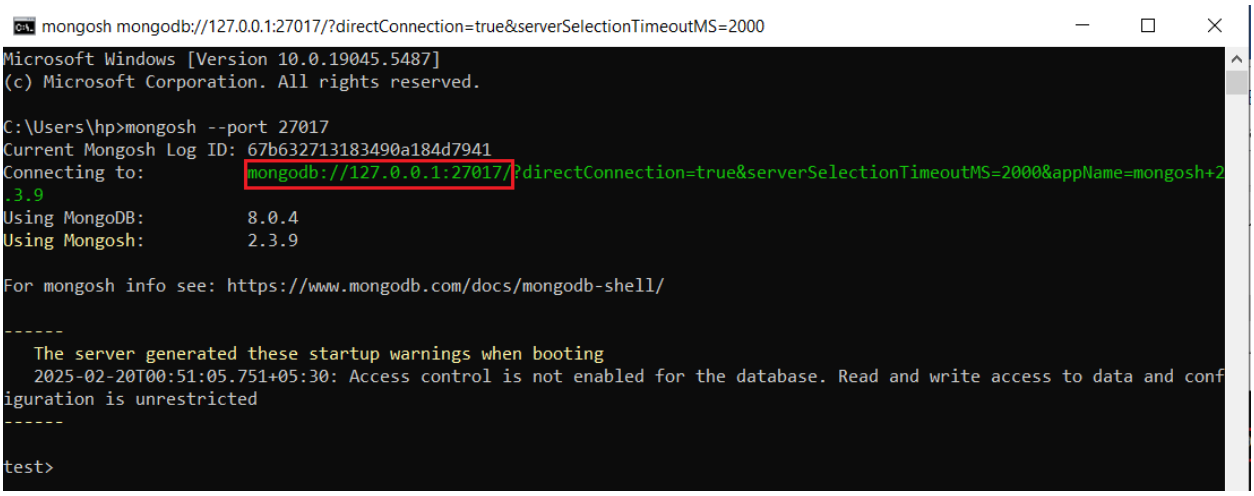


You can see that port `27020` is listening at IP address `127:0.0.4`.

9.6. Initialize Replica Set with Primary:

Open a new **Command Prompt** to execute the below command to connect to the MongoDB server instance. Here, `27017` is the default port where our MongoDB server was installed.

```
mongosh --port 27017
```



Run the following `mongosh` commands to initialize the replica set with the current instance running on localhost *(IP address `127:0.0.1`)* listening on port `27017` which is our primary instance.

```
rsconf={_id:"mongors0", members:[{_id:0,host:"localhost:27017"}]}
rs.initiate(rsconf)
```

Note that in the above replica set configuration (`rsconf` variable), we have set the `_id` to our replica set name `mongors0` and assignned `members` with `_id` as `0` and `host` as `localhost:27017`. Then, initialized replica set using `rs.initiate()` function.

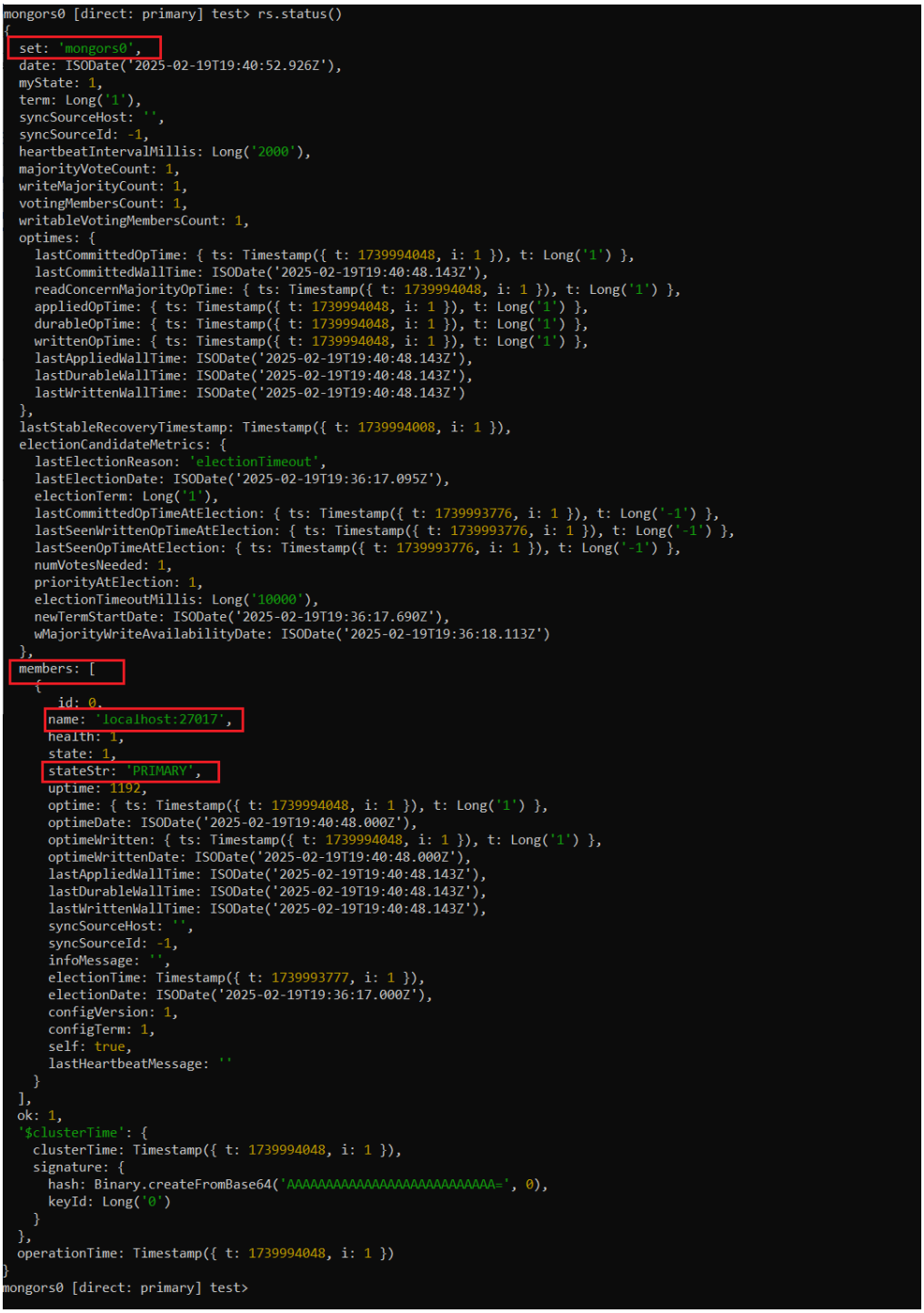
After replica set is initialized, it initially shows that the current instance is directed to secondary but, in few seconds, it is directed us to primary.

Use the following `mongosh` command to get the replica set current configuration where you can see one member running at `localhost:27017` got added.

```
rs.conf()
```

Use the following `mongosh` command to get the status of replica set where you can see the current instance `localhost:27017` is set to `PRIMARY`.

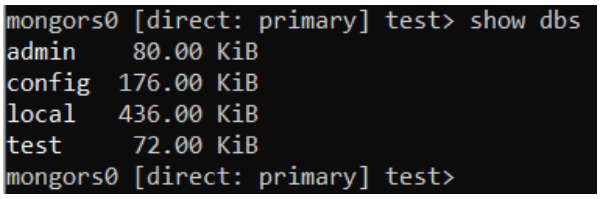
```
rs.status()
```



Now, we can run any queries to read or write data in the `mongosh` session which is connected to primary instance.

Use below `mongosh` command to get the list of current databases on the existing MongoDB server instance.

```
show dbs
```



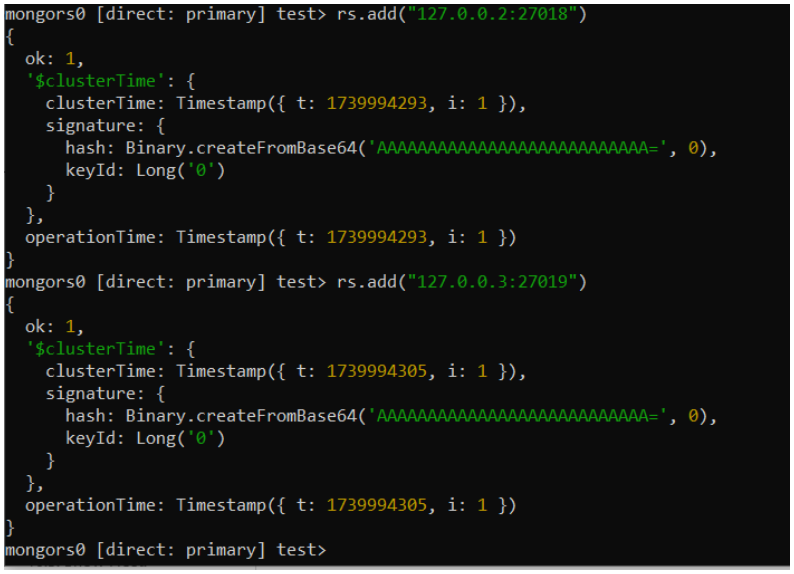
9.7. Add Secondary Members to Replica Set:

In the existing `mongosh` session *(that is connected to primary instance running on `27017` port)*, run the following commands to add our secondary instances running on hosts `127.0.0.2:27018` and `127.0.0.3:27019`. Note that you can add members only when you are connected to primary instance.

```
rs.add("127.0.0.2:27018")
rs.add("127.0.0.3:27019")
```

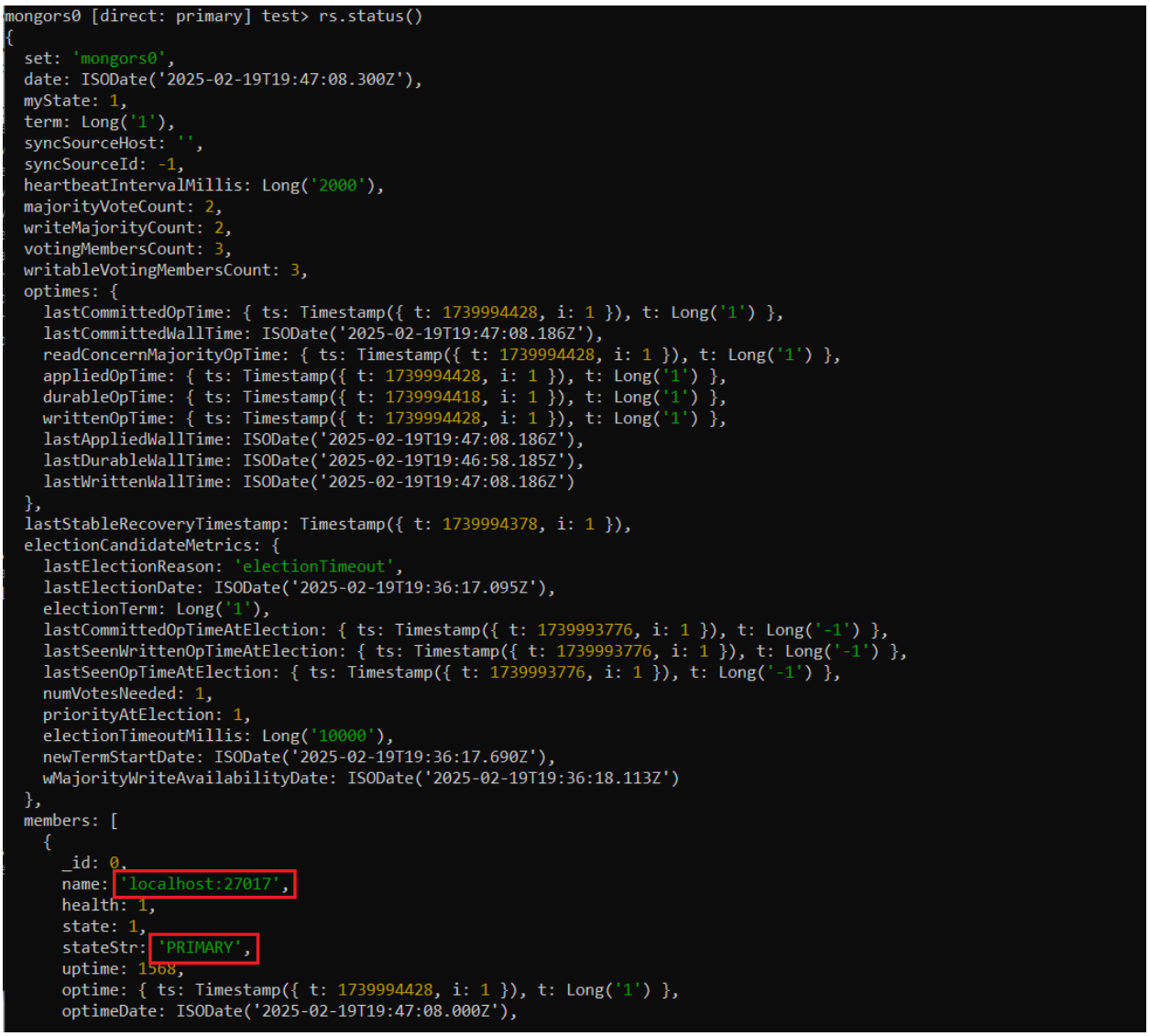
You can also use the below syntax to add members:

```
rs.add({host:"127.0.0.2:27018"})
rs.add({host:"127.0.0.3:27019"})
```



Check the status of replica set where you should see one PRIMARY and two SECONDARY members.

```
rs.status()
```



```
    optimeDate: ISODate('2025-02-19T19:47:08.000Z'),
    optimeWritten: { ts: Timestamp({ t: 1739994428, i: 1 }), t: Long('1') },
    optimeWrittenDate: ISODate('2025-02-19T19:47:08.000Z'),
    lastAppliedWallTime: ISODate('2025-02-19T19:47:08.186Z'),
    lastDurableWallTime: ISODate('2025-02-19T19:46:58.185Z'),
    lastWrittenWallTime: ISODate('2025-02-19T19:47:08.186Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1739993777, i: 1 }),
    electionDate: ISODate('2025-02-19T19:36:17.000Z'),
    configVersion: 5,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '127.0.0.2:27018',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 134,
    optime: { ts: Timestamp({ t: 1739994418, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1739994418, i: 1 }), t: Long('1') },
    optimeWritten: { ts: Timestamp({ t: 1739994418, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-02-19T19:46:58.000Z'),
    optimeDurableDate: ISODate('2025-02-19T19:46:58.000Z'),
    optimeWrittenDate: ISODate('2025-02-19T19:46:58.000Z'),
    lastAppliedWallTime: ISODate('2025-02-19T19:47:08.186Z'),
    lastDurableWallTime: ISODate('2025-02-19T19:47:08.186Z'),
    lastWrittenWallTime: ISODate('2025-02-19T19:47:08.186Z'),
    lastHeartbeat: ISODate('2025-02-19T19:47:07.634Z'),
    lastHeartbeatRecv: ISODate('2025-02-19T19:47:07.638Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27017',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 5,
    configTerm: 1
  },
  {
    _id: 2,
    name: '127.0.0.3:27019',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 122,
```

9.8. Add Arbiter Member to Replica Set:

Before adding arbiter to replica set, it is necessary to set the cluster-wide write concern, otherwise Mongo throws an error *"Reconfig attempted to install a config that would change the implicit default write concern. Use the setDefaultRWConcern command to set a cluster-wide write concern and try the reconfig again"*

```
mongors0 [direct: primary] test> rs.addArb("127.0.0.4:27020")
MongoServerError[NewReplicaSetConfigurationIncompatible]: Reconfig attempted to install a config that would change the i
mplicit default write concern. Use the setDefaultRWConcern command to set a cluster-wide write concern and try the recon
fig again.
mongors0 [direct: primary] test>
```

Write Concern describes the level of acknowledgment requested from MongoDB for write operations to either standalone Mongo or replica sets or sharded clusters. Replica sets and

sharded clusters support global (cluster-wide) default write concern which is used for operations that do not specify an explicit write concern. For more information, read this MongoDB documentation.

In the existing `mongosh` session *(that is connected to primary instance running on `27017` port)*, run the following command to set the `defaultWriteConcern` property. In this setting, `w : 2` indicates Mongo requires acknowledgement for write operations from the primary and one of secondary instances *(you can also set `w:0` requesting no acknowledgement from any member, `w:1` requesting acknowledgment from the primary alone)*

```
db.adminCommand({
   "setDefaultRWConcern" : 1,
   "defaultWriteConcern" : {
     "w" : 2
   }
 })
```

```
mongors0 [direct: primary] test> db.adminCommand({
...     "setDefaultRWConcern" : 1,
...     "defaultWriteConcern" : {
...       "w" : 2
...     }
...   })
{
  defaultReadConcern: { level: 'local' },
  defaultWriteConcern: { w: 2, wtimeout: 0 },
  updateOpTime: Timestamp({ t: 1739994588, i: 1 }),
  updateWallClockTime: ISODate('2025-02-19T19:49:48.701Z'),
  defaultWriteConcernSource: 'global',
  defaultReadConcernSource: 'implicit',
  localUpdateWallClockTime: ISODate('2025-02-19T19:49:48.892Z'),
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1739994588, i: 3 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1739994588, i: 3 })
}
mongors0 [direct: primary] test>
```

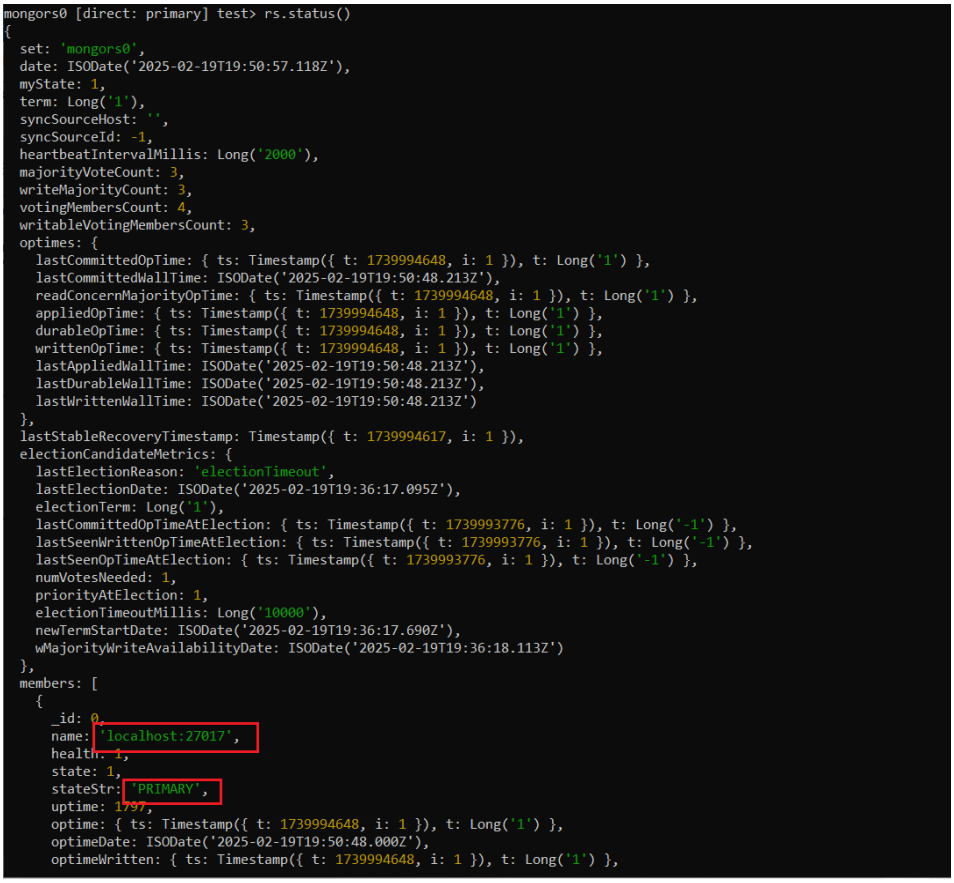
Now, run the following `mongosh` command to add the arbiter instance running on host `127.0.0.4:27020`. Note that you can add arbiter only when you are connected to primary instance.

```
rs.addArb("127.0.0.4:27020")
```

```
mongors0 [direct: primary] test> rs.addArb("127.0.0.4:27020")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1739994617, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1739994617, i: 1 })
}
mongors0 [direct: primary] test>
```

Check the status of replica set where you should see one PRIMARY, two SECONDARY and one ARBITER members.

```
rs.status()
```



```
    optimeWrittenDate: ISODate('2025-02-19T19:50:48.000Z'),
    lastAppliedWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastDurableWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastWrittenWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1739993777, i: 1 }),
    electionDate: ISODate('2025-02-19T19:36:17.000Z'),
    configVersion: 6,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '127.0.0.2:27018',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 363,
    optime: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
    optimeWritten: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-02-19T19:50:48.000Z'),
    optimeDurableDate: ISODate('2025-02-19T19:50:48.000Z'),
    optimeWrittenDate: ISODate('2025-02-19T19:50:48.000Z'),
    lastAppliedWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastDurableWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastWrittenWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastHeartbeat: ISODate('2025-02-19T19:50:55.423Z'),
    lastHeartbeatRecv: ISODate('2025-02-19T19:50:55.437Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27017',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 6,
    configTerm: 1
  },
  {
    _id: 2,
    name: '127.0.0.3:27019',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 351,
    optime: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
```

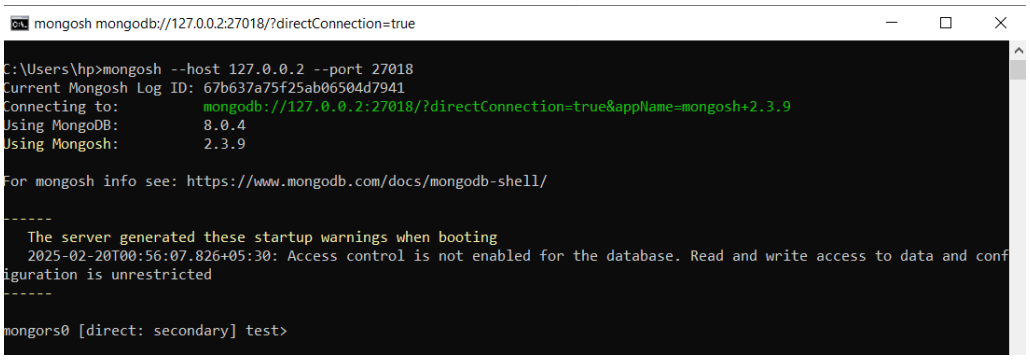
```
    optimeDurable: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
    optimeWritten: { ts: Timestamp({ t: 1739994648, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-02-19T19:50:48.000Z'),
    optimeDurableDate: ISODate('2025-02-19T19:50:48.000Z'),
    optimeWrittenDate: ISODate('2025-02-19T19:50:48.000Z'),
    lastAppliedWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastDurableWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastWrittenWallTime: ISODate('2025-02-19T19:50:48.213Z'),
    lastHeartbeat: ISODate('2025-02-19T19:50:55.472Z'),
    lastHeartbeatRecv: ISODate('2025-02-19T19:50:55.502Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27017',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 6,
    configTerm: 1
  },
  {
    _id: 3,
    name: '127.0.0.4:27020',
    health: 1,
    state: 7,
    stateStr: 'ARBITER',
    uptime: 39,
    lastHeartbeat: ISODate('2025-02-19T19:50:55.652Z'),
    lastHeartbeatRecv: ISODate('2025-02-19T19:50:55.648Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 6,
    configTerm: 1
  }
],
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1739994648, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
    keyId: Long('0')
  }
},
operationTime: Timestamp({ t: 1739994648, i: 1 })
}
mongors0 [direct: primary] test>
```

9.9. Validate Replication:

Let us verify if the replication from Primary to Secondary instances is working as expected.

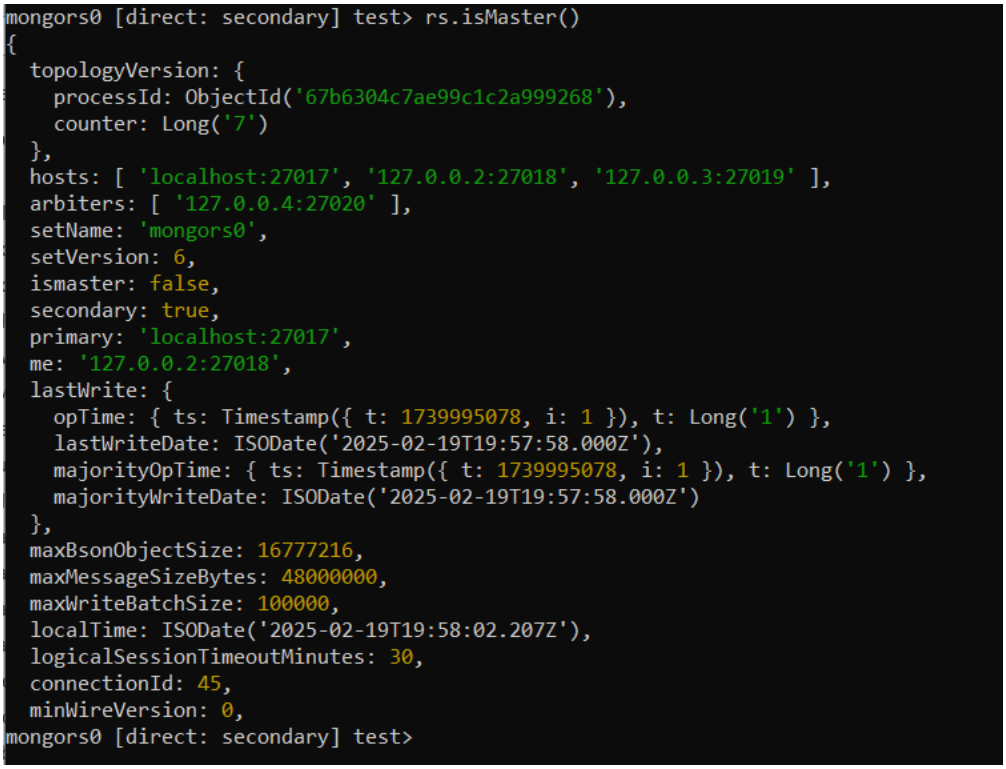
Open a new **Command Prompt** and run the following command to connect to the first secondary instance.

```
mongosh --host 127.0.0.2 --port 27018
```



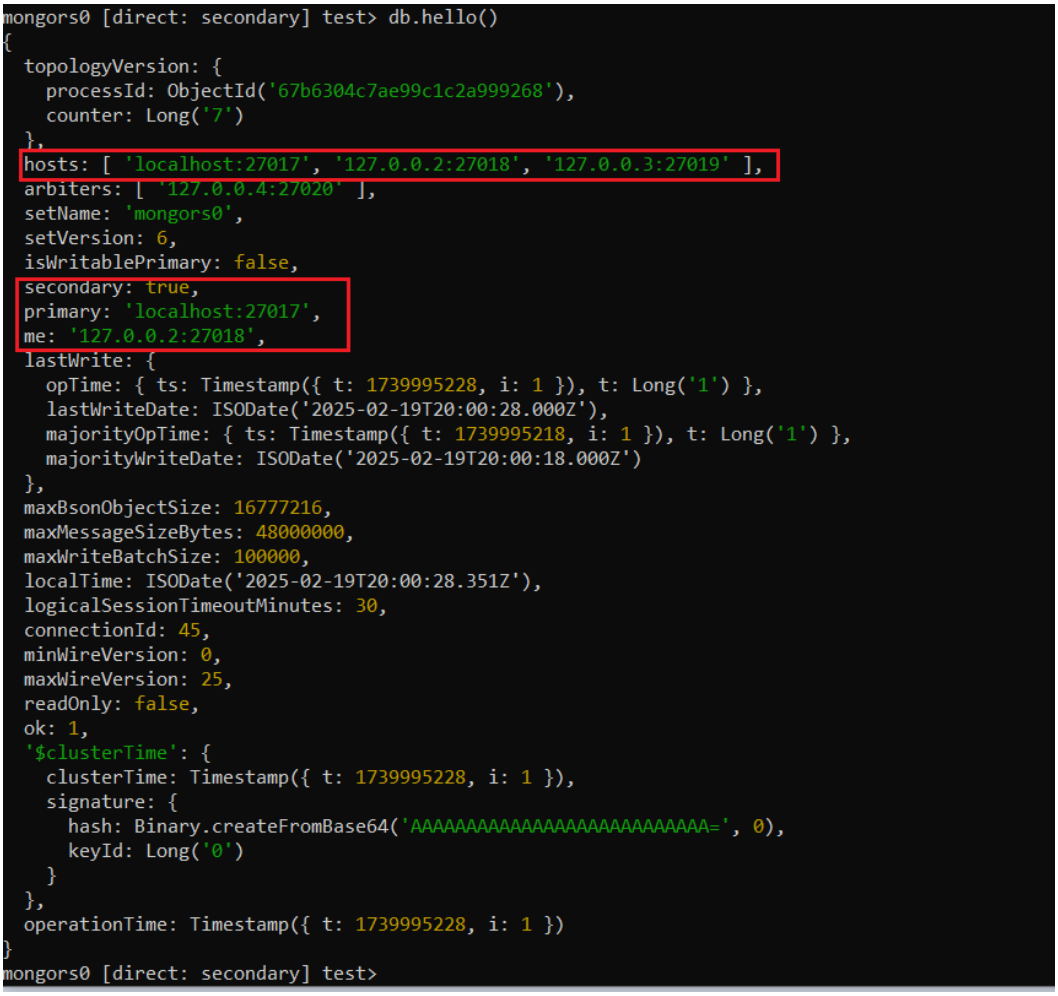
If you are not sure whether you are connected to a secondary instance, then use the below `mongosh` command which tells you if this is a primary or secondary instance.

```
rs.isMaster()
```



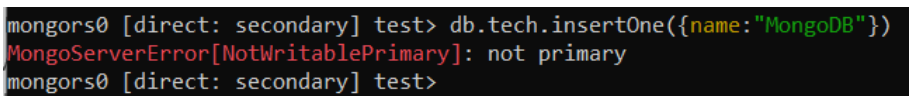
You can also use the below `mongosh` command to know which instance that you are connected to currently and what the primary is.

```
db.hello()
```



Let us try to do a write operation on the secondary instance by running the below `mongosh` command:

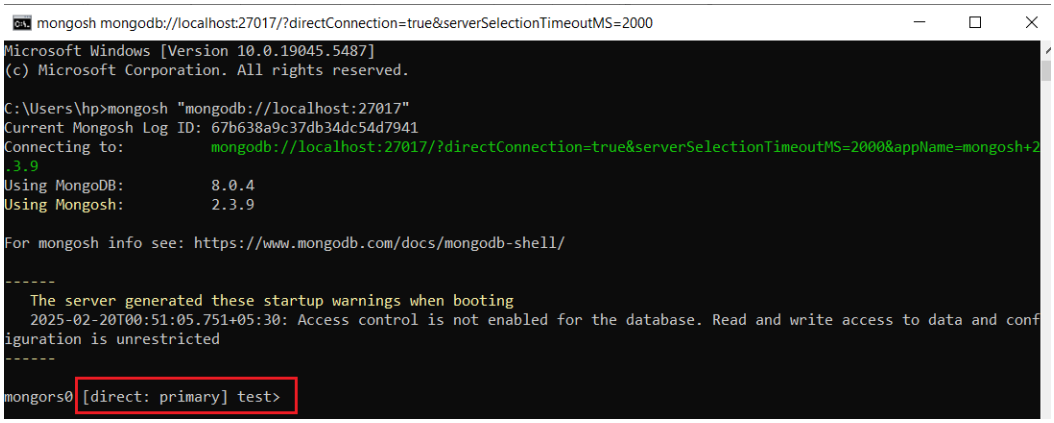
```
db.tech.insertOne({name:"MongoDB"})
```



You can see that it failed because any write operation is allowed on the primary instance only.

Open a new **Command Prompt** and connect to the primary instance as below:

```
mongosh "mongodb://localhost:27017"
```



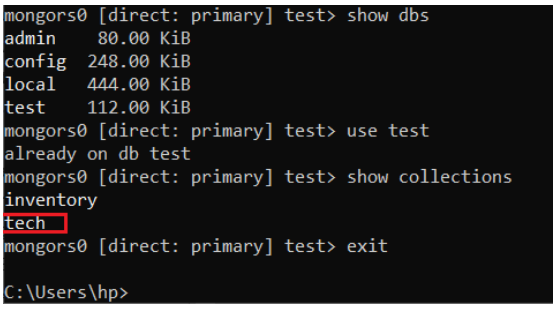
When you connect to any Mongo instance, it navigates you to `test` database by default (*though this database does not exist).*

Run the below `mongosh` command to insert a record into a collection named `tech` in the current `test` database:

```
db.tech.insertOne({name:"MongoDB"})
```

Verify if `tech` collection got created in `test` database using below `mongosh` commands:

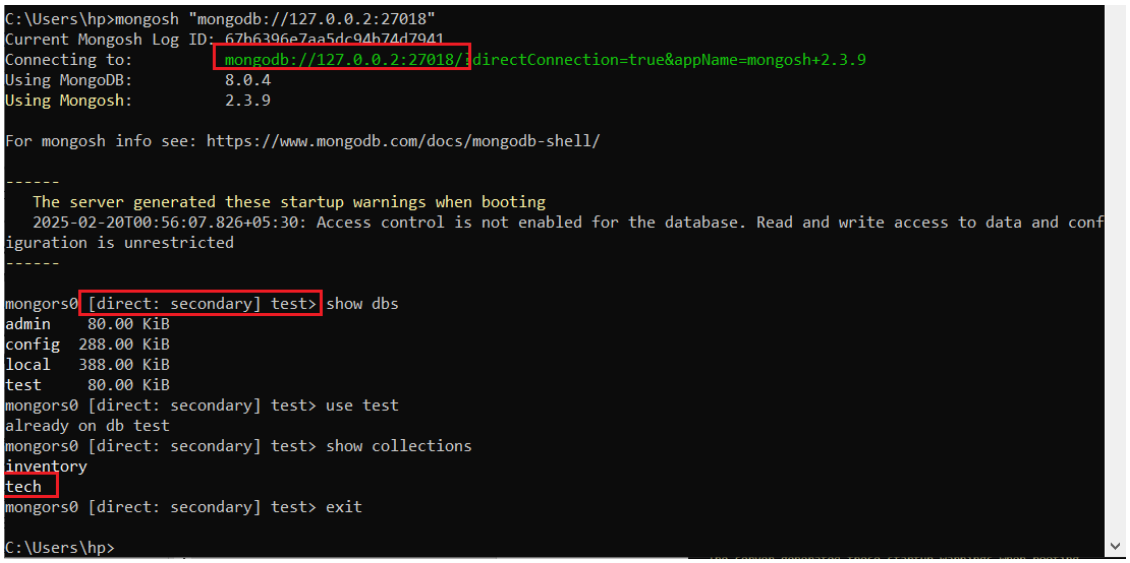
```
show dbs
use test
show collections
exit
```



Now, connect to the first secondary instance running at `127.0.0.2:27018` and verify if you can see the newly created `test` database and `tech` collection using the below commands.

```
mongosh "mongodb://127.0.0.2:27018"
```

```
show dbs
use test
show collections
exit
```



Now, connect to other secondary instance running at `127.0.0.3:27019` and verify if you can see the newly created `test` database and `tech` collection using the below commands.

```
mongosh "mongodb://127.0.0.3:27019"
```

```
show dbs
use test
show collections
exit
```

```
C:\Users\hp>mongosh "mongodb://127.0.0.3:27019"
Current Mongosh Log ID: 67b639af1b971bb2984d7941
Connecting to:          mongodb://127.0.0.3:27019/?directConnection=true&appName=mongosh+2.3.9
Using MongoDB:          8.0.4
Using Mongosh:          2.3.9

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-02-20T00:59:54.844+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

mongors0 [direct: secondary] test> show dbs
admin    80.00 KiB
config  288.00 KiB
local   388.00 KiB
test     80.00 KiB
mongors0 [direct: secondary] test> use test
already on db test
mongors0 [direct: secondary] test> show collections
inventory
tech
mongors0 [direct: secondary] test> exit

C:\Users\hp>
```

Now, let us connect to the arbiter instance running at `127.0.0.4:27020`

```
mongosh "mongodb://127.0.0.4:27020"
```

```
C:\Users\hp>mongosh "mongodb://127.0.0.4:27020"
Current Mongosh Log ID: 67b63acb363aeeec9d4d7941
Connecting to:          mongodb://127.0.0.4:27020/?directConnection=true&appName=mongosh+2.3.9
Using MongoDB:          8.0.4
Using Mongosh:          2.3.9

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-02-20T01:02:16.619+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
   2025-02-20T01:20:17.612+05:30: ** WARNING: Arbiters are not supported in quarterly binary versions
------

mongors0 [direct: arbiter] test>
```

Execute the below command and you will see an error
MongoServerError[NotPrimaryOrSecondary]: node is not in primary or recovering state

```
show dbs
```

```
mongors0 [direct: arbiter] test> show dbs
MongoServerError[NotPrimaryOrSecondary]: node is not in primary or recovering state
mongors0 [direct: arbiter] test>
```

When you try to execute any database command on the arbiter instance, it throws error since Arbiter is not meant for any data storage.

9.10. Validate Primary Election:

Let us verify if the Arbiter is able to elect Primary.

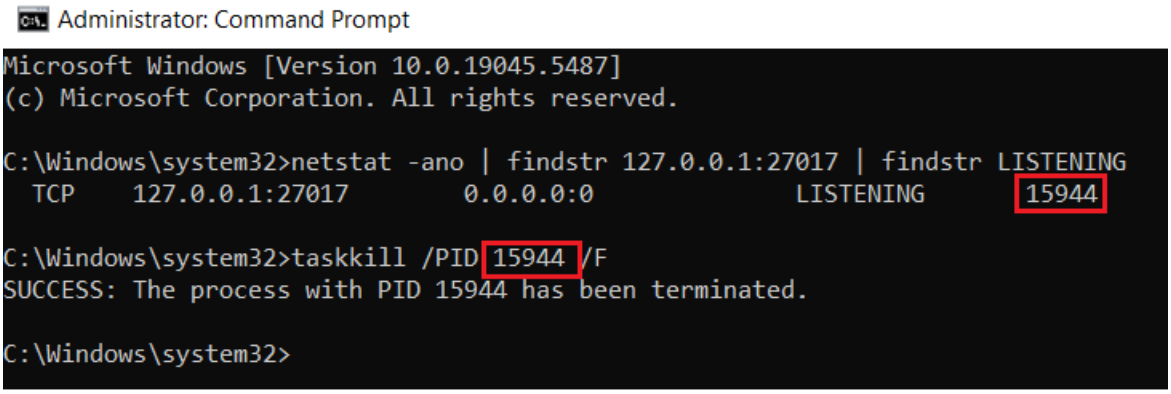
First, shutdown the primary instance.

- Open a new **Command Prompt** as **Administrator** and run the following command to find the process id where the primary instance *(127.0.0.1:27017)* is running:

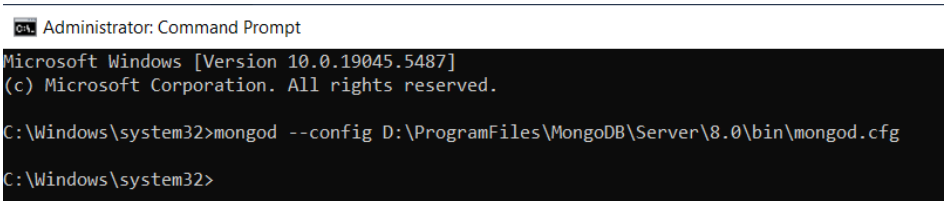
```
netstat -ano | findstr 127.0.0.1:27017 | findstr LISTENING
```

Get the process ID given by the above command and update in the below command and execute it to kill the process.

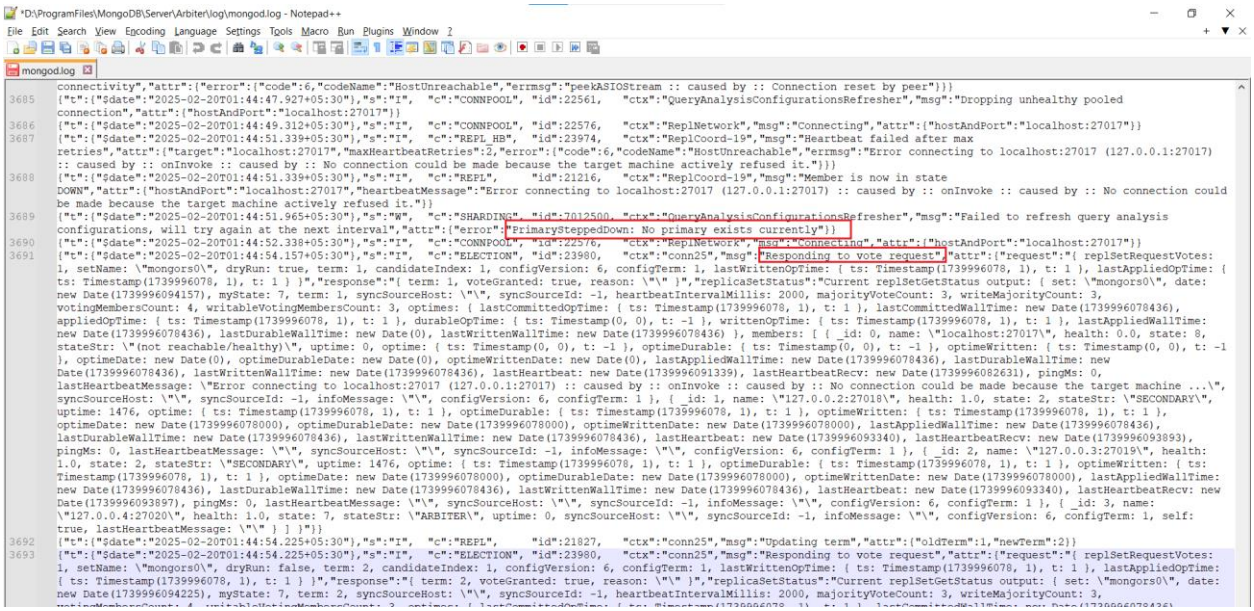
```
taskkill /PID PROCESS_ID /F
```



- You can also simply go to the **Command Prompt** where primary instance is running and press **Ctrl+C** or close the window to shut down the primary.



Now open `mongod.log` in arbiter instance at `D:\ProgramFiles\MongoDB\Server\Arbiter\log` where you should see messages such as **PrimarySteppedDown: No primary exists currently**, **Responding to vote request**, **Restarting heartbeats after learning of a new primary**, **Member is in new state, etc.** indicating that it is able to elect new primary when the existing primary is not available.



Here, you can see that host `127.0.0.2:27018` was elected as new `PRIMARY`.

10. Deploy MongoDB Sharded Cluster:

Now, we will see how to setup a MongoDB sharded cluster with the config replica set, 2 shard replica sets and the query router.

For the purpose of this documentation, we will setup the config replica set with one primary instance and one secondary instance, 2 shard replica sets each with one primary instance and two secondary instances and one query router instance running on different IP addresses and ports in the same Windows system but in the real-time project, we should configure one instance per system to setup the cluster.

10.1. Shutdown Previous MongoDB Instances:

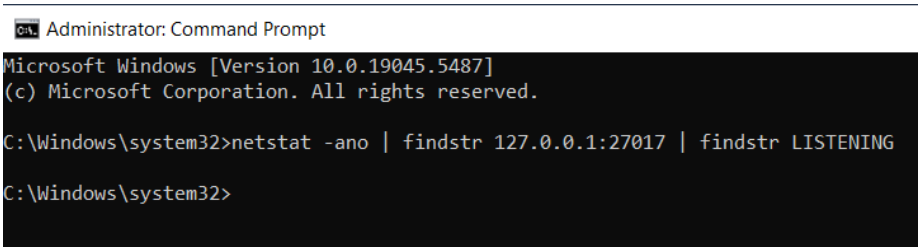
Since we have setup the replica set earlier, we should first shutdown all those instances and delete secondary and arbiter instances. If you have not setup the replica set earlier, you can skip this step.

Open a new **Command Prompt** as **Administrator** and run the following commands

- Find the process id where the primary instance *(127.0.0.1:27017)* is running and run `taskkill` command:

```
netstat -ano | findstr 127.0.0.1:27017 | findstr LISTENING
```

```
taskkill /PID PROCESS_ID /F
```

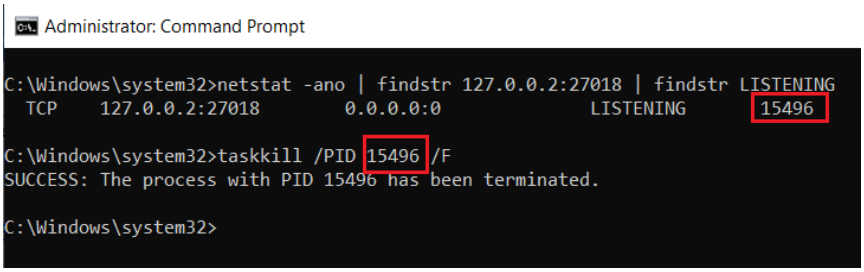


Since we have already shutdown the primary instance earlier, we do not see it running any more.

- Find the process id where the secondary1 instance *(127.0.0.2:27018)* is running and run `taskkill` command:

```
netstat -ano | findstr 127.0.0.2:27018 | findstr LISTENING
```

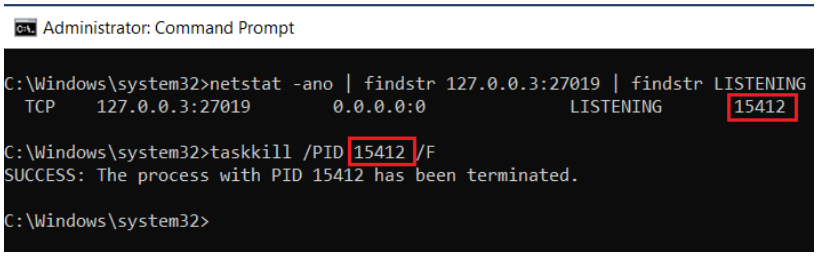
```
taskkill /PID PROCESS_ID /F
```



- Find the process id where the secondary2 instance *(127.0.0.3:27019)* is running and run `taskkill` command:

```
netstat -ano | findstr 127.0.0.3:27019 | findstr LISTENING
```

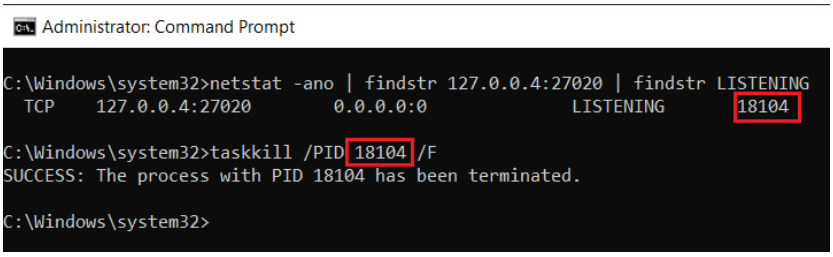
```
taskkill /PID PROCESS_ID /F
```



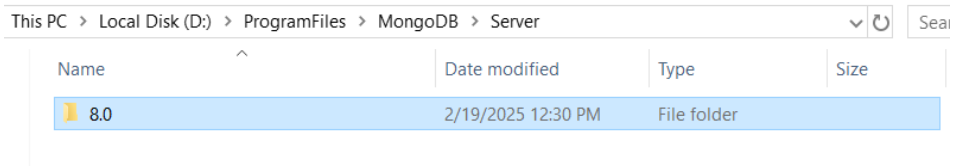
- Find the process id where the arbiter instance *(127.0.0.4:27020)* is running and run `taskkill` command:

```
netstat -ano | findstr 127.0.0.4:27020 | findstr LISTENING
```

```
taskkill /PID PROCESS_ID /F
```

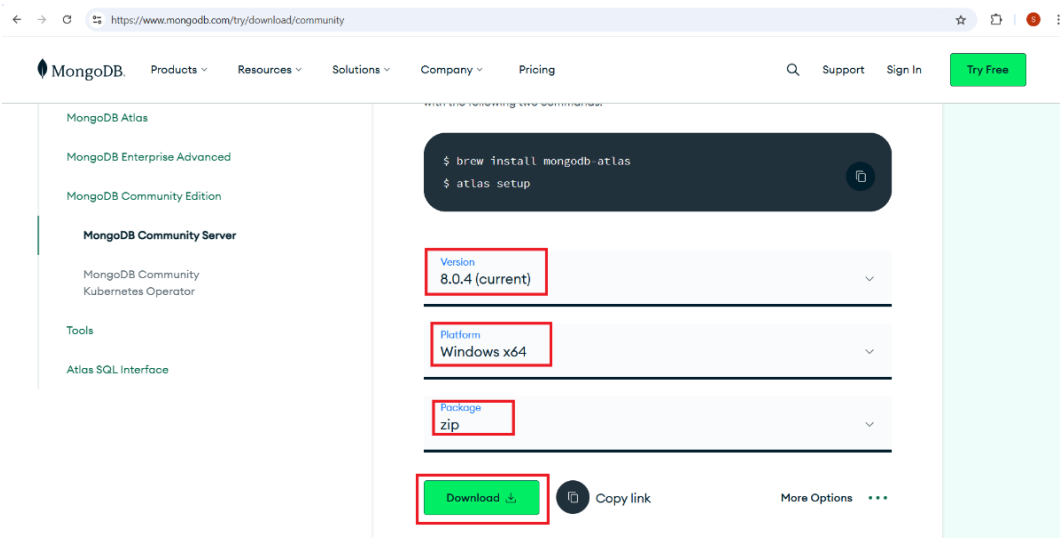


Next go to the MongoDB install path at `D:\ProgramFiles\MongoDB\Server` and remove `Arbiter`, `Secondary1` and `Seconary2` folders which we created earlier.

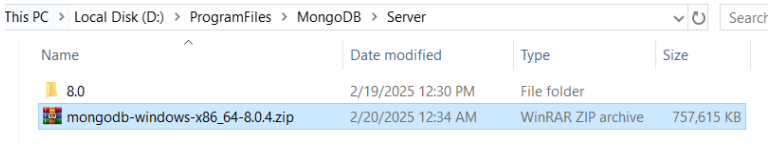


10.2. Setup Multiple Instances:

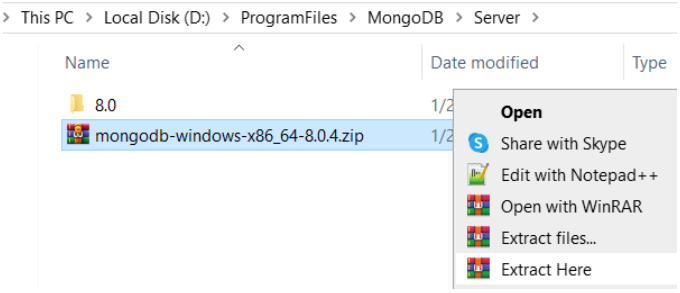
Go to [MongoDB Community Server Download Center] page. Choose the latest **Version** *(at the time of writing this document, the latest version is 8.0.4)* and **Platform** as `Windows X64` and **Package** as `zip` and click on **Download** button which downloads the file into your **Downloads** folder in your machine.



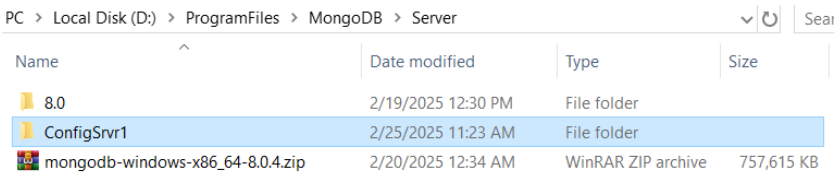
Copy the downloaded file `mongodb-windows-x86_64-8.0.4.zip` from your **Downloads** folder into MongoDB installation path at `D:\ProgramFiles\MongoDB\Server`.



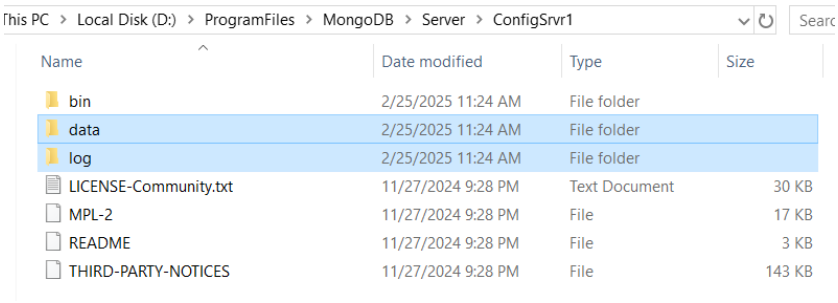
Right click on the file and select **Extract Here** option to extract contents of it into the current directory.



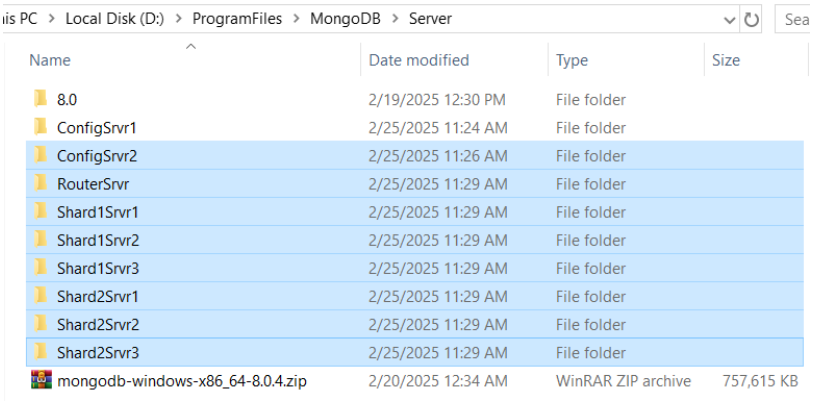
The zip file extraction may take a couple of minutes to finish. After finishing, you see a folder named `mongodb-win32-x86_64-windows-8.0.4` which consists of MongoDB binaries. Rename this newly created folder as `ConfigSrvr1`.



Create two sub-directories named `data` and `log` inside the `ConfigSrvr1` directory.



Take 8 copies of `ConfigSrvr1` folder and rename those copies as `ConfigSrvr2`, `Shard1Srvr1`, `Shard1Srvr2`, `Shard1Srvr3`, `Shard2Srvr1`, `Shard2Srvr2`, `Shard2Srvr3` and `RouterSrvr`.



10.3. Create Config Server Replica Set:

We will setup our config server instances such that they are running under different hosts and ports as below:

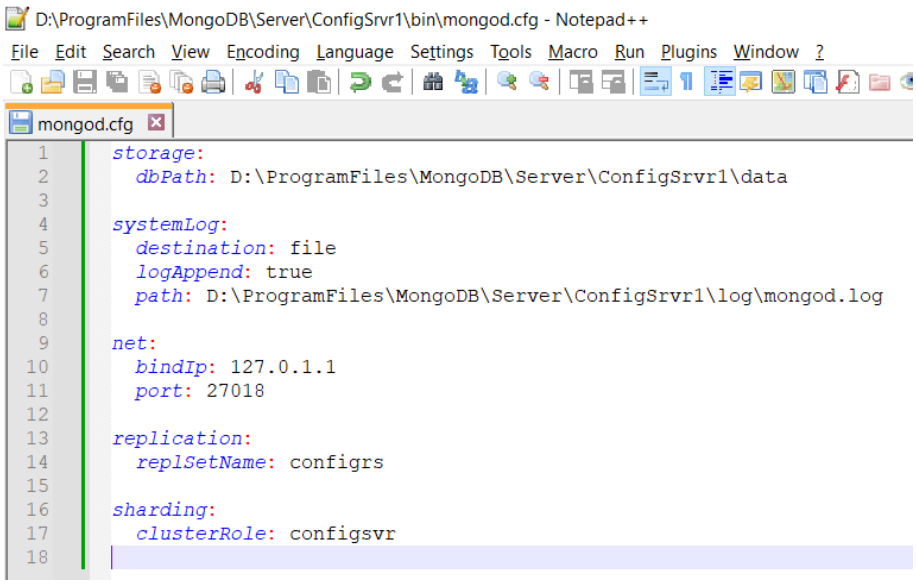
`ConfigSrvr1` instance runs at IP address `127.0.1.1` on port `27018`
`ConfigSrvr2` instance runs at IP address `127.0.1.2` on port `27018`

10.3.1. Configure Config Server Instances:

Go to `D:\ProgramFiles\MongoDB\Server\ConfigSrvr1\bin` and create `mongod.cfg` file and add the below contents into it.

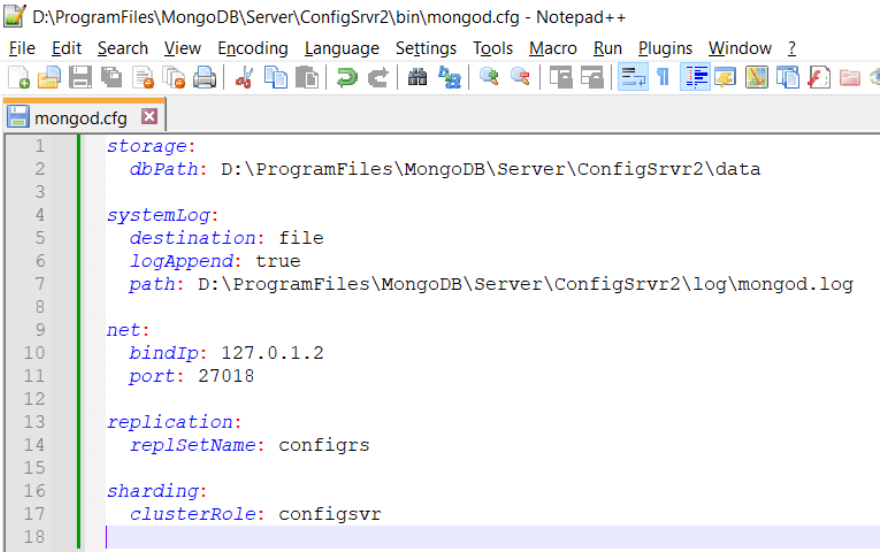
```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\ConfigSrvr1\data

systemLog:
  destination: file
  logAppend: true
```

```
  path: D:\ProgramFiles\MongoDB\Server\ConfigSrvr1\log\mongod.log

net:
  bindIp: 127.0.1.1
  port: 27018

replication:
  replSetName: configrs

sharding:
  clusterRole: configsvr
```



Similarly, go to `D:\ProgramFiles\MongoDB\Server\ConfigSrvr2\bin` and create `mongod.cfg` file and add the below contents into it.

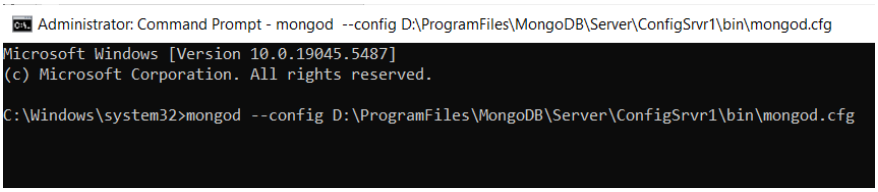
```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\ConfigSrvr2\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\ConfigSrvr2\log\mongod.log

net:
  bindIp: 127.0.1.2
  port: 27018

replication:
  replSetName: configrs

sharding:
  clusterRole: configsvr
```



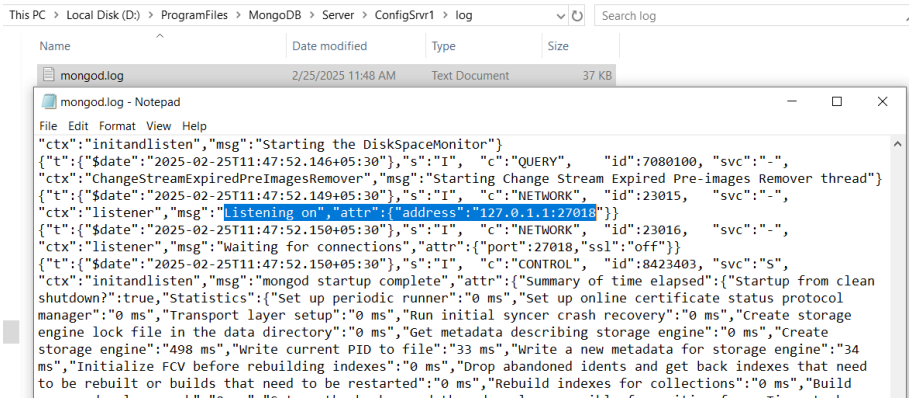
10.3.2. Start Config Server Instances:

Open a new **Command Prompt** application as **Administrator** and execute the below command to start `ConfigSrvr1` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\ConfigSrvr1\bin\mongod.cfg
```

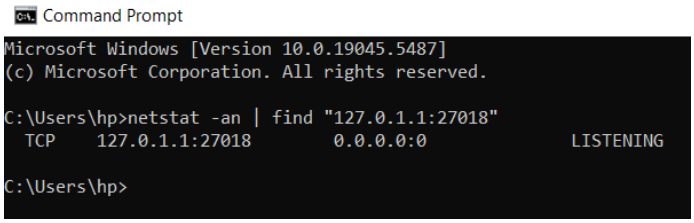


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\ConfigSrvr1\log` and look for messages **"Listening on address 127.0.1.1:27018"** which indicate that MongoDB instance has been started successfully.



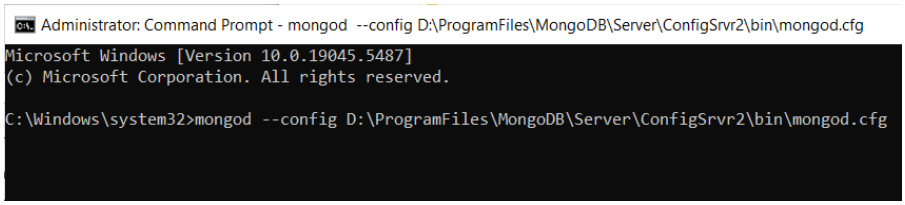
You can also verify if the address `127.0.1.1:27018` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.1.1:27018"
```

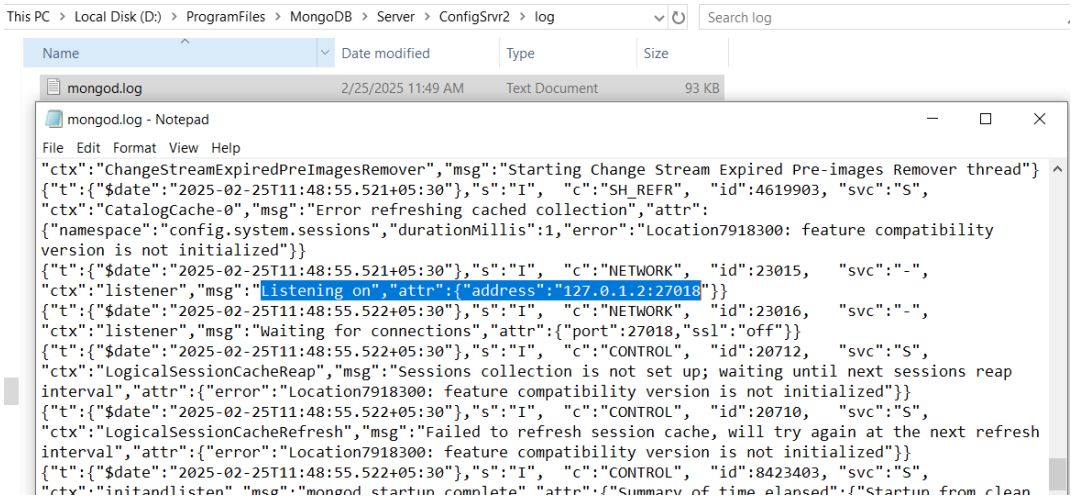


Open a new **Command Prompt** application as **Administrator** and execute the below command to start the `ConfigSrvr2` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\ConfigSrvr2\bin\mongod.cfg
```

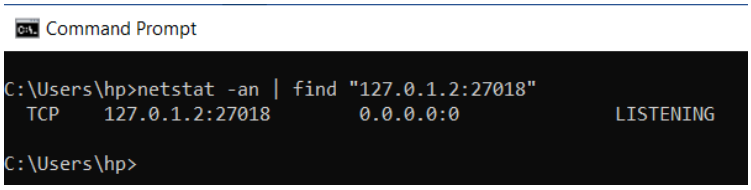


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\ConfigSrvr2\log` and look for messages **"Listening on address 127.0.1.2:27018"** which indicate that MongoDB instance has been started successfully.



You can also verify if the address `127.0.1.2:27018` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.1.2:27018"
```

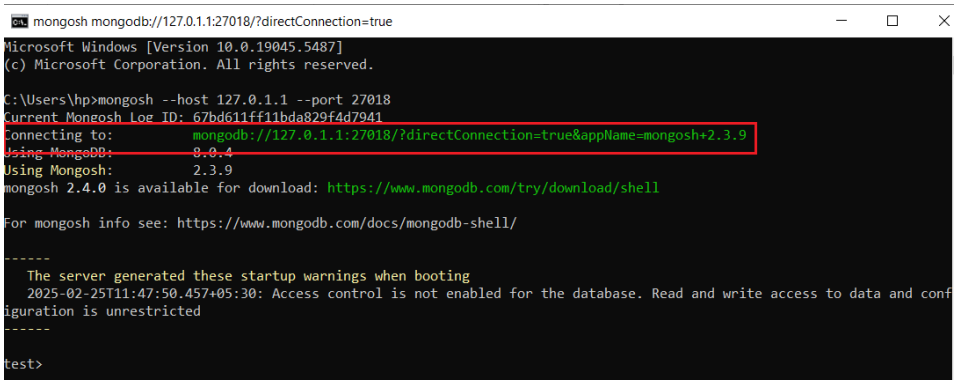


10.3.3. Initiate Config Server Replica Set:

We can connect to any config server instance to initialize the config server replica set

Open a new **Command Prompt** and execute the below command to connect to the `ConfigSrvr1` MongoDB server instance listening at `127.0.1.1:27018` address.

```
mongosh --host 127.0.1.1 --port 27018
```



When you are connected successfully, it will give you `test>` prompt to access the `test` database by default.

On the `test>` prompt, run the following command to initialize the replica set with 2 config servers running at `127.0.1.1:27018` and `127.0.1.2:27018`. Make sure that the replica set name given in the `_id` field below must match with the `replSetName` value in the `D:\ProgramFiles\MongoDB\Server\ConfigSrvr1\mongod.cfg` file.

```
rs.initiate(
  {
    _id : "configrs",
    members: [
      { _id : 0, host : "127.0.1.1:27018" },
```

```
      { _id : 1, host : "127.0.1.2:27018" }
    ]
  }
)
```

```
test> rs.initiate(
...   {
...     _id : "configrs",
...     members: [
...       { _id : 0, host : "127.0.1.1:27018" },
...       { _id : 1, host : "127.0.1.2:27018" }
...     ]
...   }
... )
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1740464465, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1740464465, i: 1 })
}
configrs [direct: secondary] test>

configrs [direct: secondary] test>

configrs [direct: secondary] test>

configrs [direct: secondary] test>

configrs [direct: secondary] test>

configrs [direct: secondary] test>

configrs [direct: secondary] test>

configrs [direct: primary] test>
```

If the replica set is initiated successfully, `zrs.initiate()` method returns `ok` as `1` and provides the `clusterTime` details. It initially directs the current instance to secondary but in few seconds, it directs to primary indicating that the current instance is set to primary.

Run the following command to get the replica set current configuration where you can see 2 members running at `127.0.1.1:27018` and `127.0.1.2:27018` got added to the replica set:

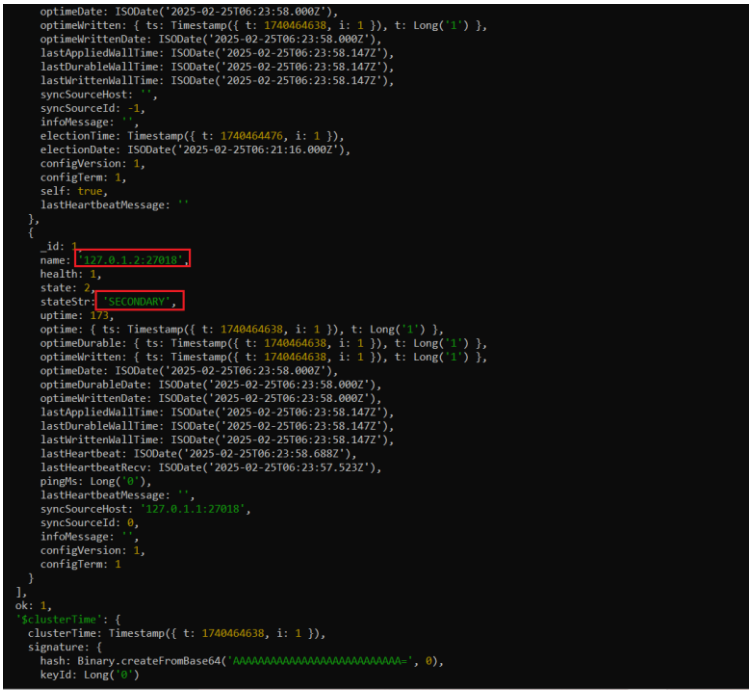
```
rs.conf()
```

```
configrs [direct: primary] test> rs.conf()
{
  _id: 'configrs',
  version: 1,
  term: 1,
  members: [
    {
      id: 0,
      host: '127.0.1.1:27018',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long('0'),
      votes: 1
    },
    {
      _id: 1,
      host: '127.0.1.2:27018',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long('0'),
      votes: 1
    }
  ],
  configsvr: true,
  protocolVersion: Long('1'),
  writeConcernMajorityJournalDefault: true,
  settings: {
    chainingAllowed: true,
    heartbeatIntervalMillis: 2000,
    heartbeatTimeoutSecs: 10,
    electionTimeoutMillis: 10000,
    catchUpTimeoutMillis: -1,
    catchUpTakeoverDelayMillis: 30000,
    getLastErrorModes: {},
    getLastErrorDefaults: { w: 1, wtimeout: 0 },
    replicaSetId: ObjectId('67bd6150253f4b463d90beb6')
  }
}
configrs [direct: primary] test>
```

Run the following command to get the status of replica set where you can see the current instance `127.0.1.1:27018` is set to `PRIMARY` and the other instance `127.0.1.2:27018` is set to `SECONDARY`.

```
rs.status()
```

```
configrs [direct: primary] test> rs.status()
{
  set: 'configrs',
  date: ISODate('2025-02-25T06:23:59.055Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  configsvr: true,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 2,
  writableVotingMembersCount: 2,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1740464638, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2025-02-25T06:23:58.147Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1740464638, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1740464638, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1740464638, i: 1 }), t: Long('1') },
    writtenOpTime: { ts: Timestamp({ t: 1740464638, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2025-02-25T06:23:58.147Z'),
    lastDurableWallTime: ISODate('2025-02-25T06:23:58.147Z'),
    lastWrittenWallTime: ISODate('2025-02-25T06:23:58.147Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1740464586, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate('2025-02-25T06:21:16.190Z'),
    electionTerm: Long('1'),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1740464465, i: 1 }), t: Long('-1') },
    lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1740464465, i: 1 }), t: Long('-1') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1740464465, i: 1 }), t: Long('-1') },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long('10000'),
    numCatchUpOps: Long('0'),
    newTermStartDate: ISODate('2025-02-25T06:21:16.679Z'),
    wMajorityWriteAvailabilityDate: ISODate('2025-02-25T06:21:18.783Z')
  },
  members: [
    {
      _id: 0,
      name: '127.0.1.1:27018',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 373,
      optime: { ts: Timestamp({ t: 1740464638, i: 1 }), t: Long('1') },
```



10.4. Create Shard Clusters:

We will setup 2 shard clusters (replica sets) such that they are running under different hosts and ports as below:

Shard Replica Set1:

`Shard1Srvr1` instance runs at IP address `127.0.2.1` on port `27019`
`Shard1Srvr2` instance runs at IP address `127.0.2.2` on port `27019`
`Shard1Srvr3` instance runs at IP address `127.0.2.3` on port `27019`

Shard Replica Set2:

`Shard2Srvr1` instance runs at IP address `127.0.3.1` on port `27019`
`Shard2Srvr2` instance runs at IP address `127.0.3.2` on port `27019`
`Shard2Srvr3` instance runs at IP address `127.0.3.3` on port `27019`

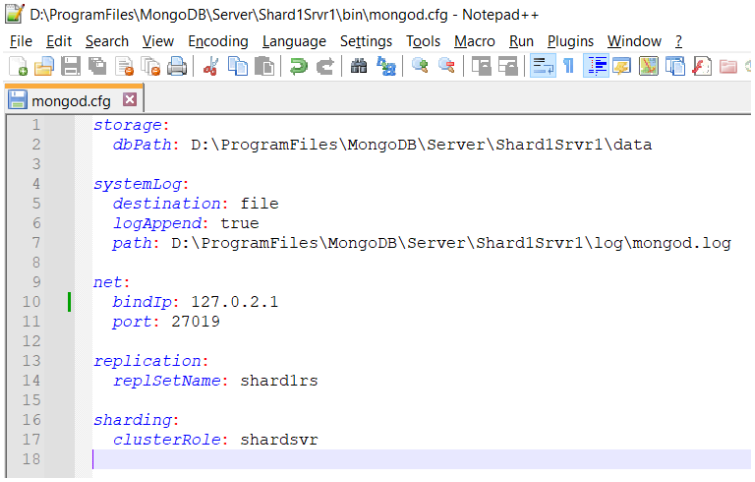
Note: Ensure that no process is running at `127.0.2.1:27019`, `127.0.2.2:27019`, `127.0.2.3:27019`, `127.0.3.1:27019`, `127.0.3.2:27019`, and `127.0.3.3:27019` addresses in your Windows system.

10.4.1. Configure Shard Server Instances:

Follow the below steps to setup the first shard replica set:

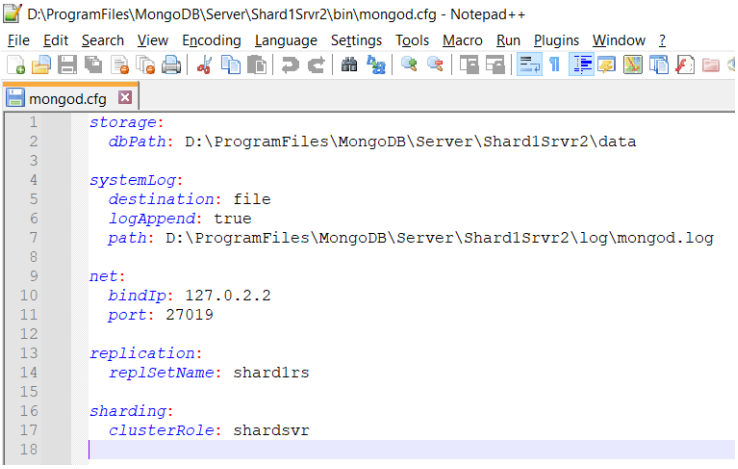
- Go to `D:\ProgramFiles\MongoDB\Server\Shard1Srvr1\bin` and create `mongod.cfg` file and add the below contents into it.

```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Shard1Srvr1\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\Shard1Srvr1\log\mongod.log

net:
  bindIp: 127.0.2.1
  port: 27019

replication:
  replSetName: shard1rs

sharding:
  clusterRole: shardsvr
```



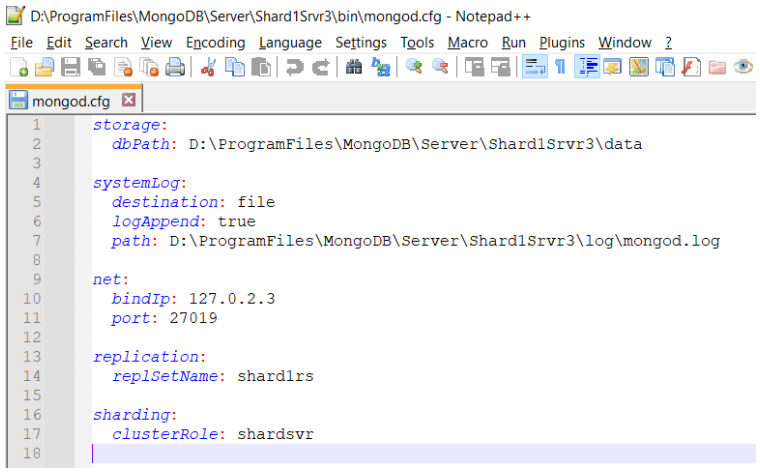
- Similarly, go to `D:\ProgramFiles\MongoDB\Server\Shard1Srvr2\bin` and create `mongod.cfg` file and add the below contents into it.

```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Shard1Srvr2\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\Shard1Srvr2\log\mongod.log

net:
  bindIp: 127.0.2.2
  port: 27019

replication:
  replSetName: shard1rs

sharding:
  clusterRole: shardsvr
```



- Similarly, go to `D:\ProgramFiles\MongoDB\Server\Shard1Srvr3\bin` and create `mongod.cfg` file and add the below contents into it.

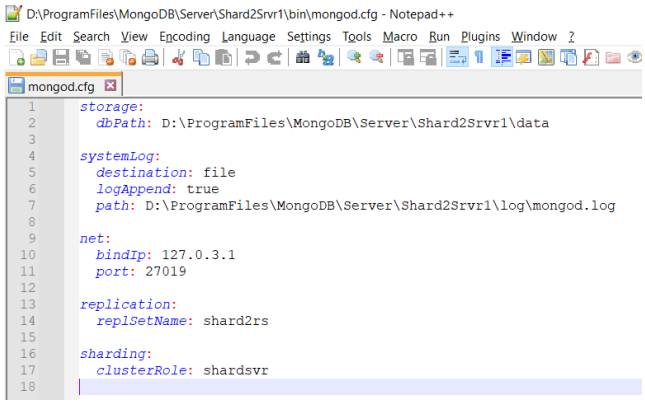
```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Shard1Srvr3\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\Shard1Srvr3\log\mongod.log

net:
  bindIp: 127.0.2.3
  port: 27019

replication:
  replSetName: shard1rs

sharding:
  clusterRole: shardsvr
```



Follow the below steps to setup the second shard replica set:

- Go to `D:\ProgramFiles\MongoDB\Server\Shard2Srvr1\bin` and create `mongod.cfg` file and add the below contents into it.

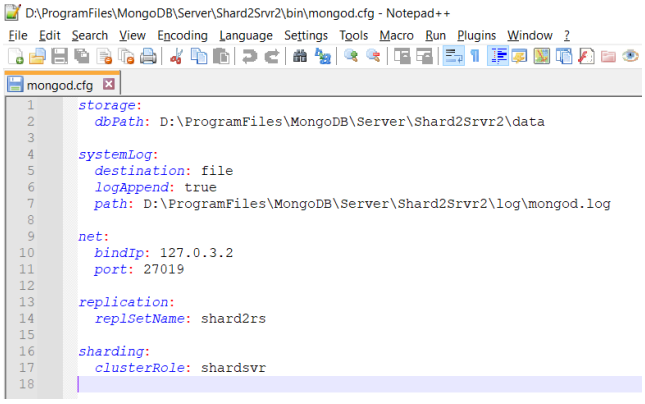
```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Shard2Srvr1\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\Shard2Srvr1\log\mongod.log

net:
  bindIp: 127.0.3.1
  port: 27019

replication:
  replSetName: shard2rs

sharding:
  clusterRole: shardsvr
```



- Similarly, go to `D:\ProgramFiles\MongoDB\Server\Shard2Srvr2\bin` and create `mongod.cfg` file and add the below contents into it.

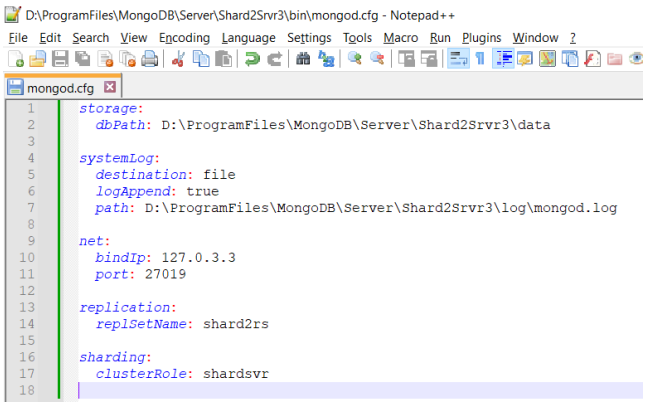
```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Shard2Srvr2\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\Shard2Srvr2\log\mongod.log

net:
  bindIp: 127.0.3.2
  port: 27019
```

```
replication:
  replSetName: shard2rs

sharding:
  clusterRole: shardsvr
```



- Similarly, go to `D:\ProgramFiles\MongoDB\Server\Shard2Srvr3\bin` and create `mongod.cfg` file and add the below contents into it.

```
storage:
  dbPath: D:\ProgramFiles\MongoDB\Server\Shard2Srvr3\data

systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\Shard2Srvr3\log\mongod.log

net:
  bindIp: 127.0.3.3
  port: 27019

replication:
  replSetName: shard2rs

sharding:
  clusterRole: shardsvr
```

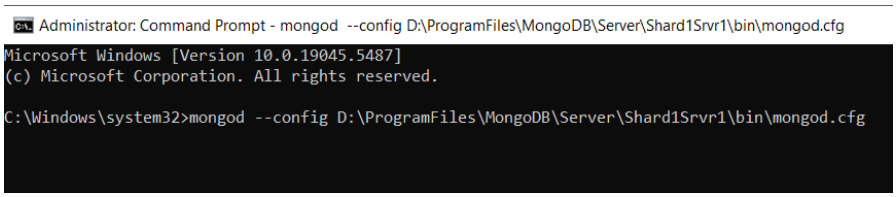


10.4.2. Start Shard Server Instances:

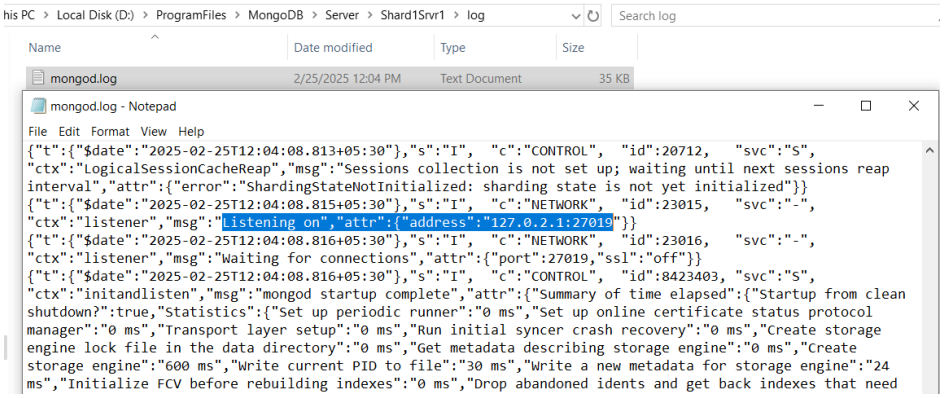
Now, let us start all shard server instances.

Open a new **Command Prompt** application as **Administrator** and execute the below command to start `Shard1Srvr1` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Shard1Srvr1\bin\mongod.cfg
```

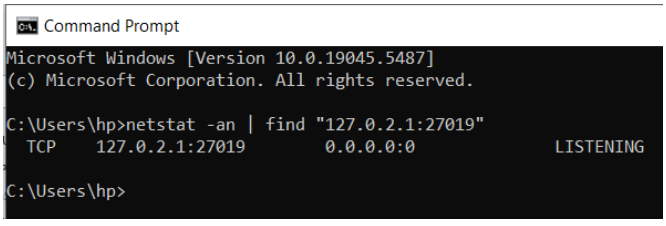


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Shard1Srvr1\log` and look for messages **"Listening on address 127.0.2.1:27019"** which indicate that MongoDB instance has been started successfully.



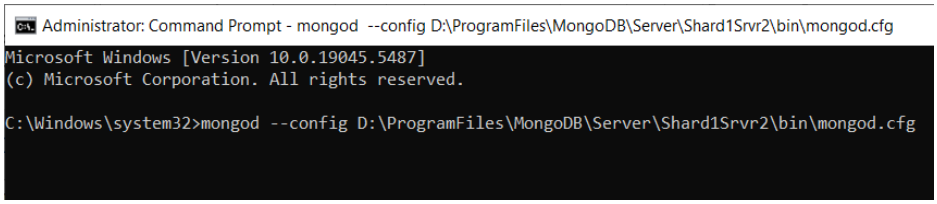
You can also verify if the address `127.0.2.1:27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.2.1:27019"
```

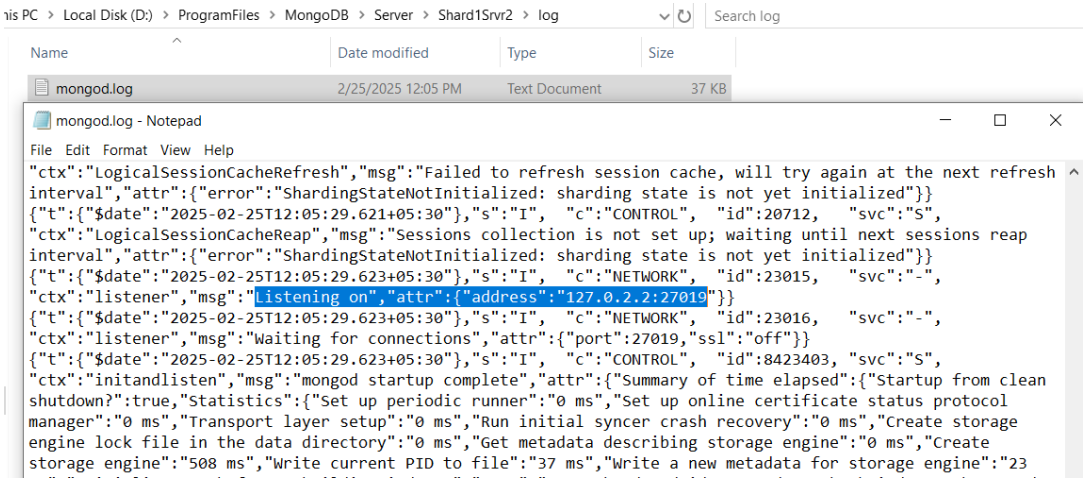


Open a new **Command Prompt** application as **Administrator** and execute the below command to start `Shard1Srvr2` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Shard1Srvr2\bin\mongod.cfg
```

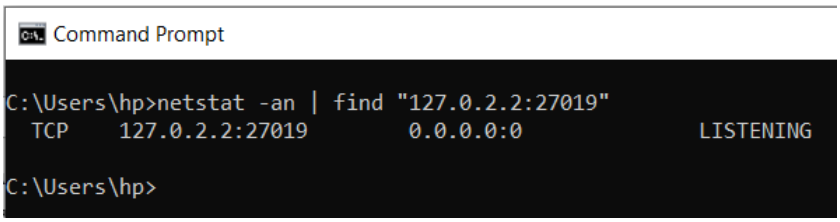


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Shard1Srvr2\log` and look for messages **"Listening on address 127.0.2.2:27019"** which indicate that MongoDB instance has been started successfully.



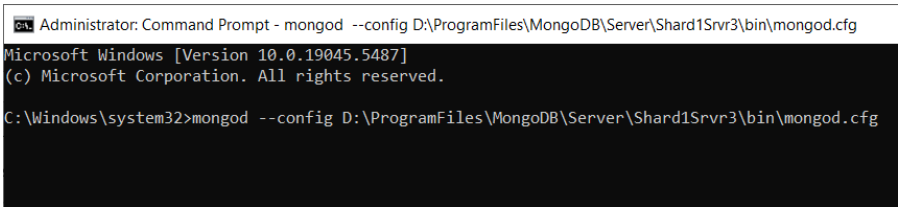
You can also verify if the address `127.0.2.2:27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.2.2:27019"
```

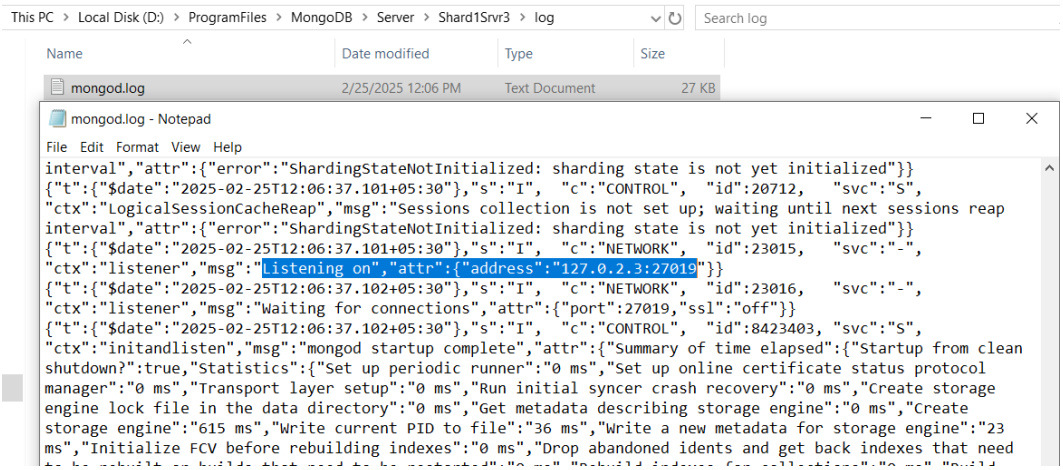


Open a new **Command Prompt** application as **Administrator** and execute the below command to start `Shard1Srvr3` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Shard1Srvr3\bin\mongod.cfg
```

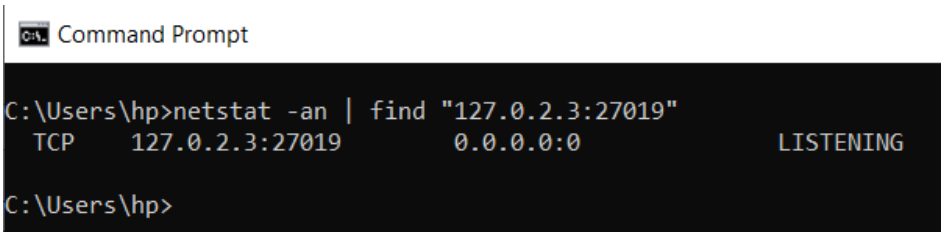


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Shard1Srvr3\log` and look for messages **"Listening on address 127.0.2.3:27019"** which indicate that MongoDB instance has been started successfully.



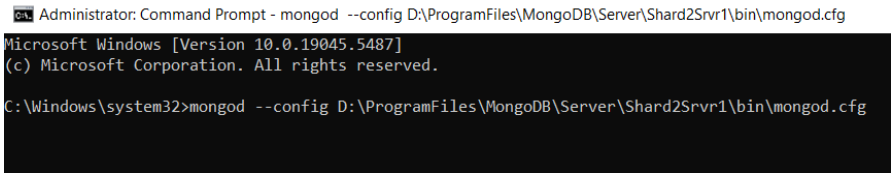
You can also verify if the address `127.0.2.3:27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.2.3:27019"
```



Open a new **Command Prompt** application as **Administrator** and execute the below command to start `Shard2Srvr1` instance.

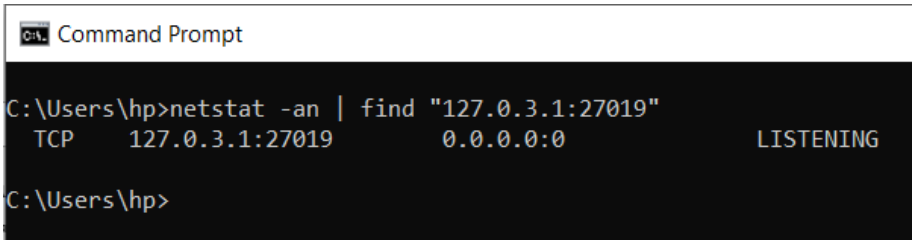
```
mongod --config
D:\ProgramFiles\MongoDB\Server\Shard2Srvr1\bin\mongod.cfg
```



Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Shard2Srvr1\log` and look for messages **"Listening on address 127.0.3.1:27019"** which indicate that MongoDB instance has been started successfully.

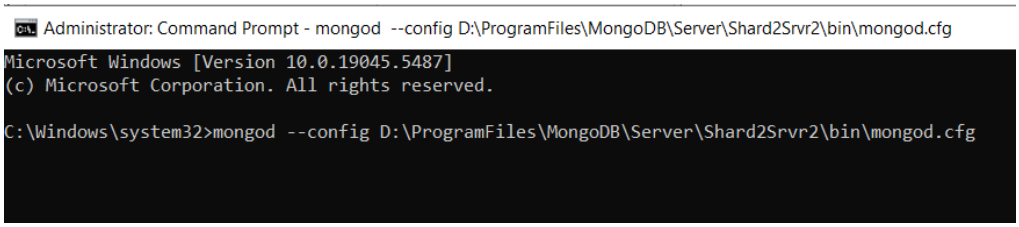
You can also verify if the address `127.0.3.1:27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.3.1:27019"
```

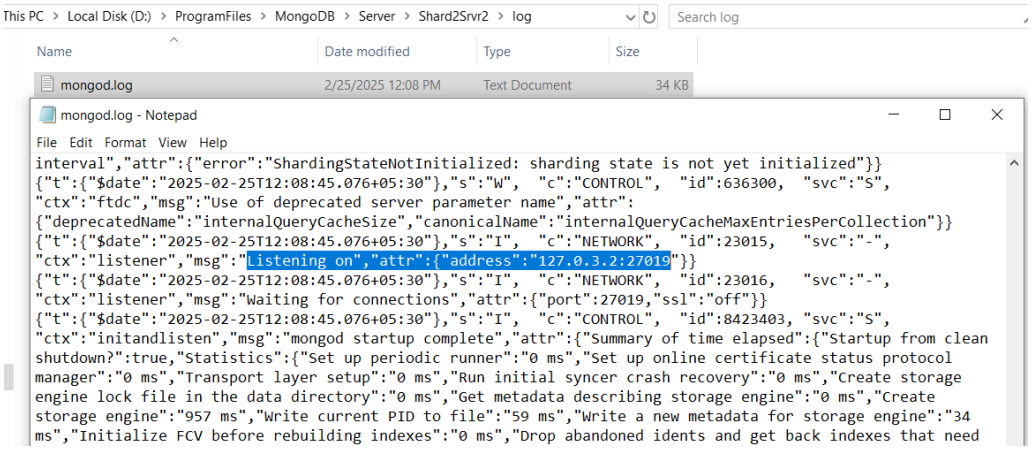


Open a new **Command Prompt** application as **Administrator** and execute the below command to start `Shard2Srvr2` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Shard2Srvr2\bin\mongod.cfg
```

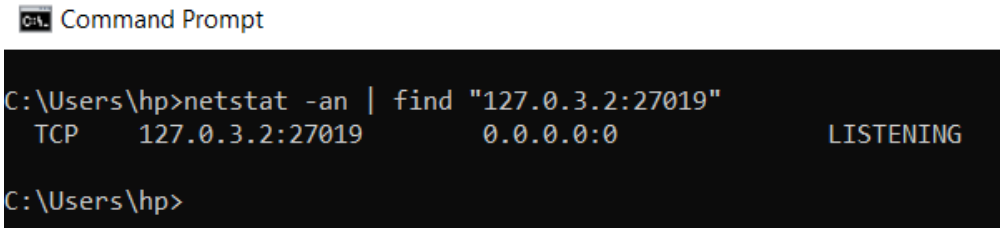


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Shard2Srvr2\log` and look for messages **"Listening on address 127.0.3.2:27019"** which indicate that MongoDB instance has been started successfully.



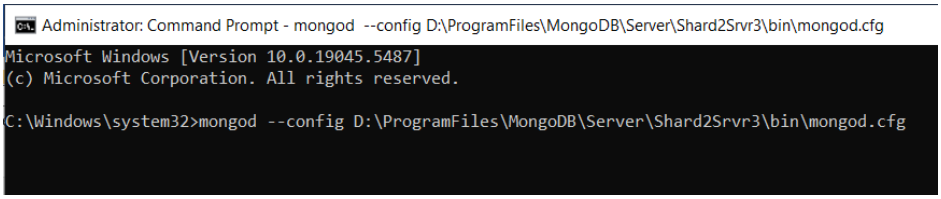
You can also verify if the address `127.0.3.2:27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.3.2:27019"
```

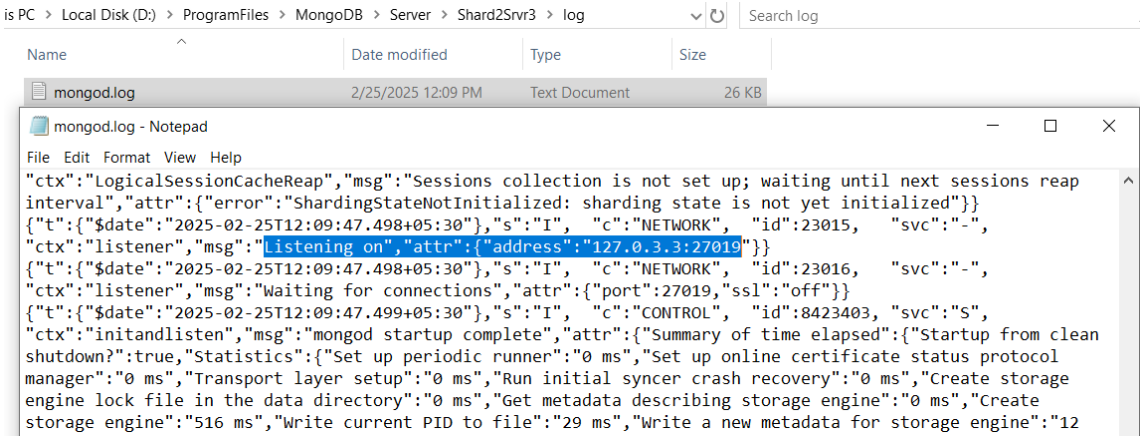


Open a new **Command Prompt** application as **Administrator** and execute the below command to start `Shard2Srvr3` instance.

```
mongod --config
D:\ProgramFiles\MongoDB\Server\Shard2Srvr3\bin\mongod.cfg
```

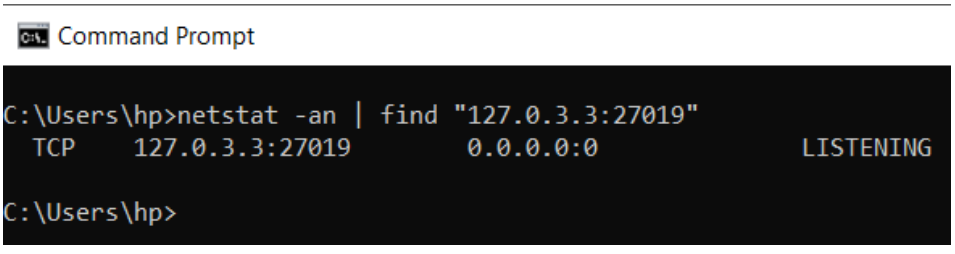


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\Shard2Srvr3\log` and look for messages **"Listening on address 127.0.3.3:27019"** which indicate that MongoDB instance has been started successfully.



You can also verify if the address `127.0.3.3:27019` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.3.3:27019"
```

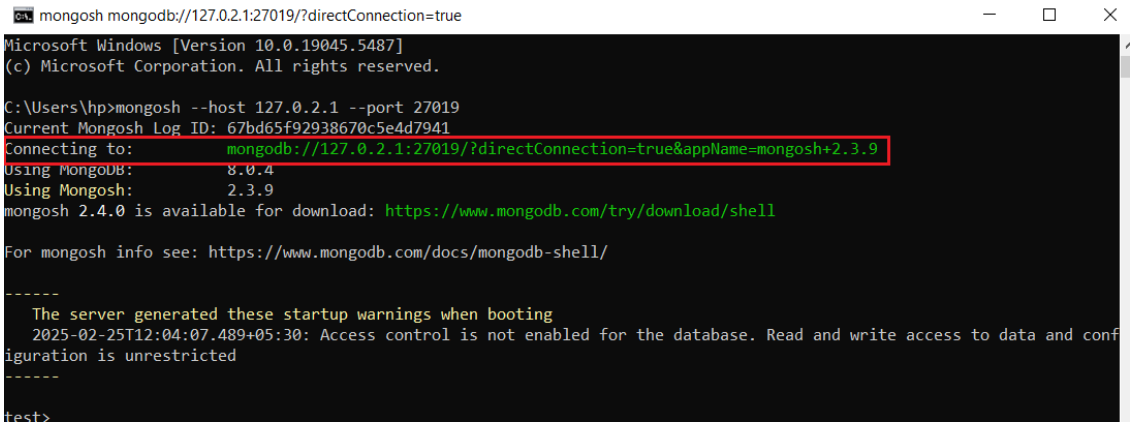


10.4.3. Initiate Shard Server Replica Sets:

Follow the below steps to setup the first shard replica set:

- Open a new **Command Prompt** and execute the below command to connect to the `Shard1Srvr1` MongoDB server instance listening at `127.0.2.1:27019` address.

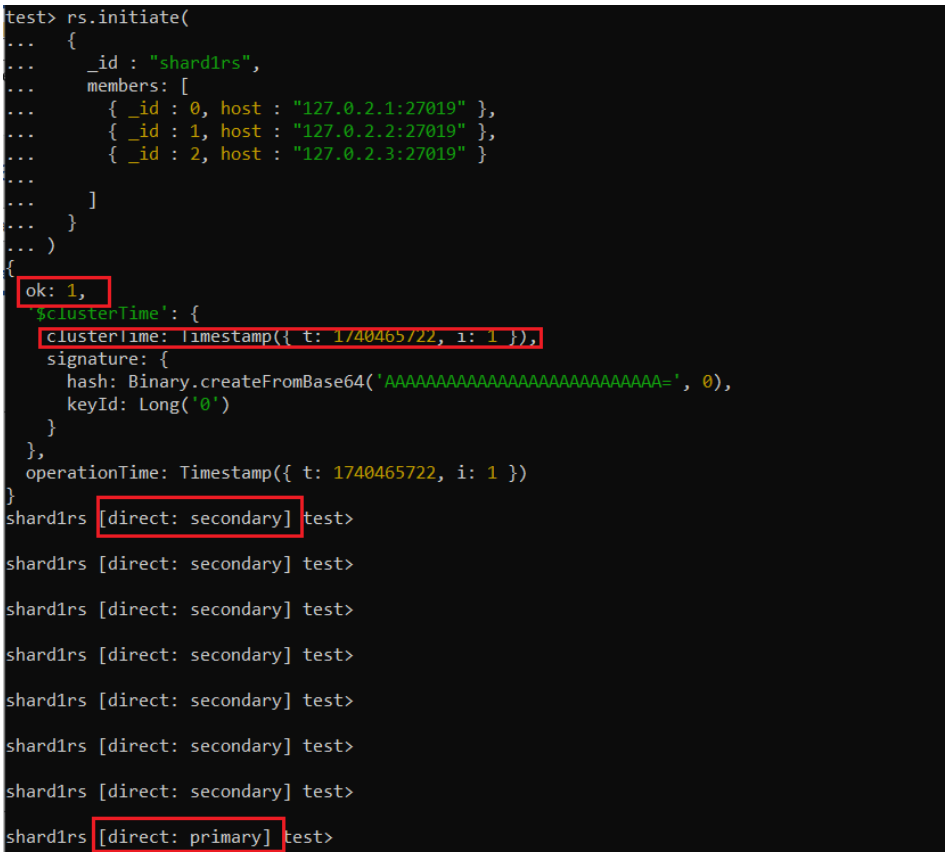
```
mongosh --host 127.0.2.1 --port 27019
```



When you are connected successfully, it will give you `test>` prompt to access the `test` database by default.

On the `test>` prompt, run the following command to initialize the replica set with 3 shard servers running at `127.0.2.1:27019`, `127.0.2.2:27019` and `127.0.2.3:27019`. Make sure that the replica set name given in the `_id` field below must match with the `replSetName` value in the `D:\ProgramFiles\MongoDB\Server\Shard1Srvr1\mongod.cfg` file.

```
rs.initiate(
  {
    _id : "shard1rs",
    members: [
      { _id : 0, host : "127.0.2.1:27019" },
      { _id : 1, host : "127.0.2.2:27019" },
      { _id : 2, host : "127.0.2.3:27019" }

    ]
  }
)
```



If the replica set is initiated successfully, the `rs.initiate()` method returns the `ok` as `1` and provides the `clusterTime` details. It initially directs the current instance to secondary but in few seconds, it directs to primary indicating that the current instance is set to primary.

Run the following command to get the replica set current configuration where you can see 2 members running at `127.0.2.1:27019`, `127.0.2.2:27019` and `127.0.2.3:27019` got added to the replica set.

```
rs.conf()
```

```
shard1rs [direct: primary] test> rs.conf()
{
  _id: 'shard1rs',
  version: 1,
  term: 1,
  members: [
    {
      _id: 0,
      host: '127.0.2.1:27019',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long('0'),
      votes: 1
    },
    {
      _id: 1,
      host: '127.0.2.2:27019',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long('0'),
      votes: 1
    },
    {
      _id: 2,
      host: '127.0.2.3:27019',
      arbiterOnly: false,
      buildIndexes: true,
      hidden: false,
      priority: 1,
      tags: {},
      secondaryDelaySecs: Long('0'),
      votes: 1
    }
  ],
  protocolVersion: Long('1'),
  writeConcernMajorityJournalDefault: true,
  settings: {
    chainingAllowed: true,
    heartbeatIntervalMillis: 2000,
    heartbeatTimeoutSecs: 10,
    electionTimeoutMillis: 10000,
    catchUpTimeoutMillis: -1,
    catchUpTakeoverDelayMillis: 30000,
```

Run the following command to get the status of replica set where you can see the current instance `127.0.3.1:27019` is set to `PRIMARY` and the other 2 instances `127.0.2.2:27019` and `127.0.2.3:27019` are set to `SECONDARY`.

```
rs.status()
```

```
    optimeWritten: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    optimeWrittenDate: ISODate('2025-02-25T06:44:24.000Z'),
    lastAppliedWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastDurableWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastWrittenWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1740465733, i: 1 }),
    electionDate: ISODate('2025-02-25T06:42:13.000Z'),
    configVersion: 1,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: '127.0.2.2:27019',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 142,
    optime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    optimeDurable: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    optimeWritten: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    optimeDate: ISODate('2025-02-25T06:44:24.000Z'),
    optimeDurableDate: ISODate('2025-02-25T06:44:24.000Z'),
    optimeWrittenDate: ISODate('2025-02-25T06:44:24.000Z'),
    lastAppliedWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastDurableWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastWrittenWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastHeartbeat: ISODate('2025-02-25T06:44:26.106Z'),
    lastHeartbeatRecv: ISODate('2025-02-25T06:44:24.988Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: '127.0.2.1:27019',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 1,
    configTerm: 1
  },
  {
    _id: 2,
    name: '127.0.2.3:27019',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 142,
    optime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
```

```
shard1rs [direct: primary] test> rs.status()
{
  set: 'shard1rs',
  date: ISODate('2025-02-25T06:44:26.140Z'),
  myState: 1,
  term: Long('1'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('2000'),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    lastCommittedWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    appliedOpTime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    durableOpTime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    writtenOpTime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
    lastAppliedWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastDurableWallTime: ISODate('2025-02-25T06:44:24.616Z'),
    lastWrittenWallTime: ISODate('2025-02-25T06:44:24.616Z')
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1740465834, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate('2025-02-25T06:42:13.592Z'),
    electionTerm: Long('1'),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1740465722, i: 1 }), t: Long('-1') },
    lastSeenWrittenOpTimeAtElection: { ts: Timestamp({ t: 1740465722, i: 1 }), t: Long('-1') },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1740465722, i: 1 }), t: Long('-1') },
    numVotesNeeded: 2,
    priorityAtElection: 1,
    electionTimeoutMillis: Long('10000'),
    numCatchUpOps: Long('0'),
    newTermStartDate: ISODate('2025-02-25T06:42:14.127Z'),
    wMajorityWriteAvailabilityDate: ISODate('2025-02-25T06:42:14.627Z')
  },
  members: [
    {
      _id: 0,
      name: '127.0.2.1:27019',
      health: 1,
      state: 1,
      stateStr: 'PRIMARY',
      uptime: 621,
      optime: { ts: Timestamp({ t: 1740465864, i: 1 }), t: Long('1') },
      optimeDate: ISODate('2025-02-25T06:44:24.000Z'),
```

Follow the below steps to setup the second shard replica set:

- Open a new **Command Prompt** and execute the below command to connect to the `Shard1Srvr1` MongoDB server instance listening at `127.0.3.1:27019` address.

```
mongosh --host 127.0.3.1 --port 27019
```

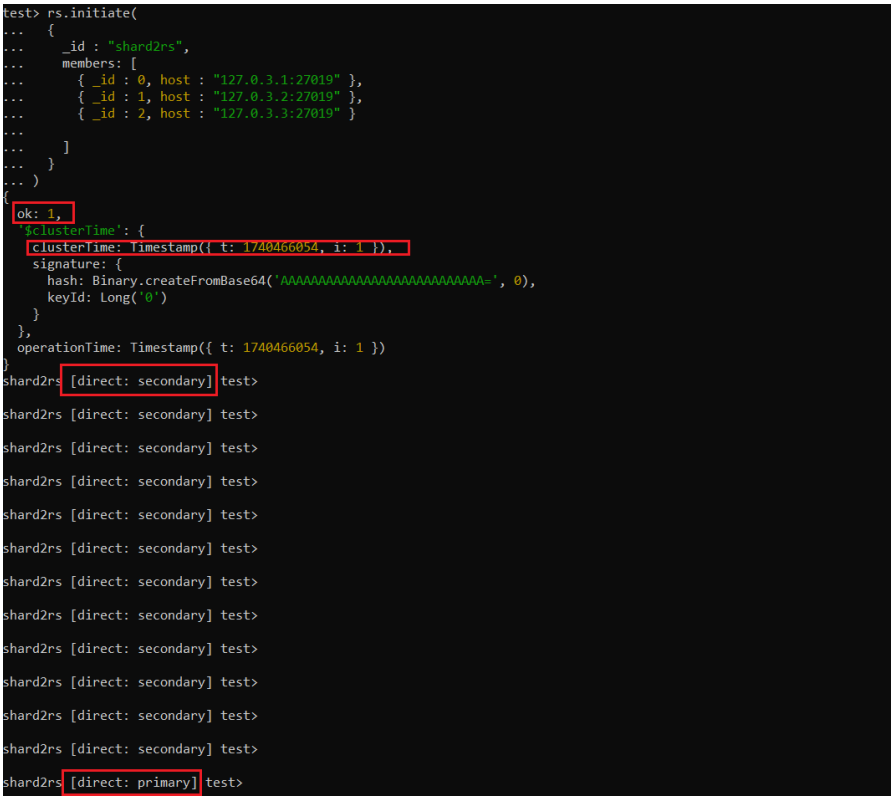
```
mongosh mongodb://127.0.3.1:27019/?directConnection=true
Microsoft Windows [Version 10.0.19045.5487]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>mongosh --host 127.0.3.1 --port 27019
Current Mongosh Log ID: 67bd6749877c7018b24d7941
Connecting to:          mongodb://127.0.3.1:27019/?directConnection=true&appName=mongosh+2.3.9
Using MongoDB:          8.0.4
Using Mongosh:          2.3.9
mongosh 2.4.0 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-02-25T12:07:42.578+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------

test>
```

When you are connected successfully, it will give you `test>` prompt to access the `test` database by default.

On the `test>` prompt, run the following command to initialize the replica set with 3 shard servers running at `127.0.3.1:27019`, `127.0.3.2:27019` and `127.0.3.3:27019`. Make sure that the replica set name given in the `_id` field below

must match with the `replSetName` value in the `D:\ProgramFiles\MongoDB\Server\Shard2Srvr1\mongod.cfg` file.

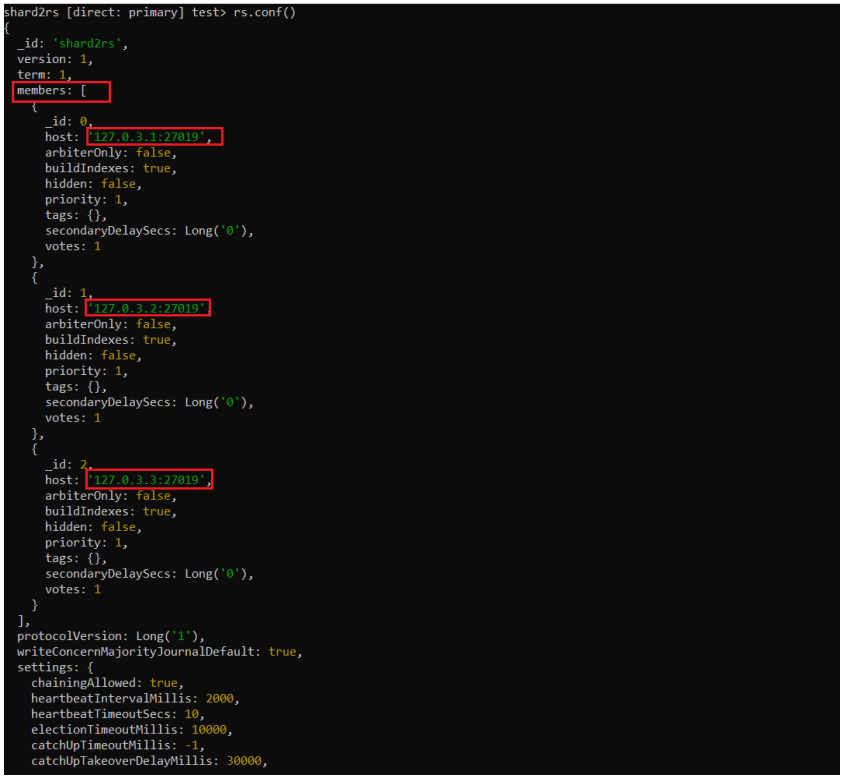
```
rs.initiate(
  {
    _id : "shard2rs",
    members: [
      { _id : 0, host : "127.0.3.1:27019" },
      { _id : 1, host : "127.0.3.2:27019" },
      { _id : 2, host : "127.0.3.3:27019" }

    ]
  }
)
```



If the replica set is initiated successfully, the `rs.initiate()` method returns the `ok` as `1` and provides the `clusterTime` details. It then initially directs the current instance to secondary but in few seconds, it directs to primary indicating that the current instance is set to primary.

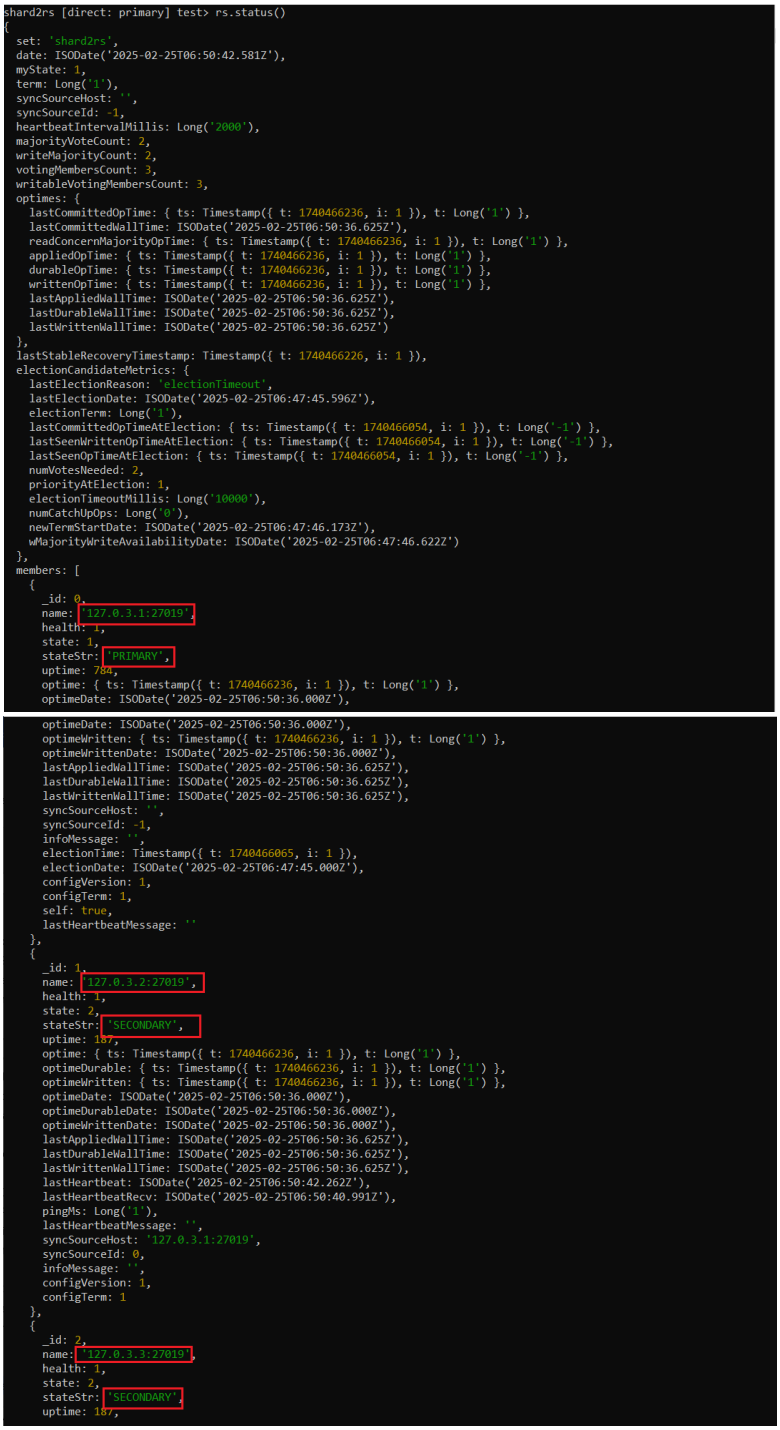
Run the following command to get the replica set current configuration where you can see 3 members running at `127.0.3.1:27019`, `127.0.3.2:27019` and `127.0.3.3:27019` got added to the replica set.

```
rs.conf()
```



Run the following command to get the status of replica set where you can see the current instance `127.0.3.1:27019` is set to `PRIMARY` and the other 2 instances `127.0.3.2:27019` and `127.0.3.3:27019` are set to `SECONDARY`.

```
rs.status()
```



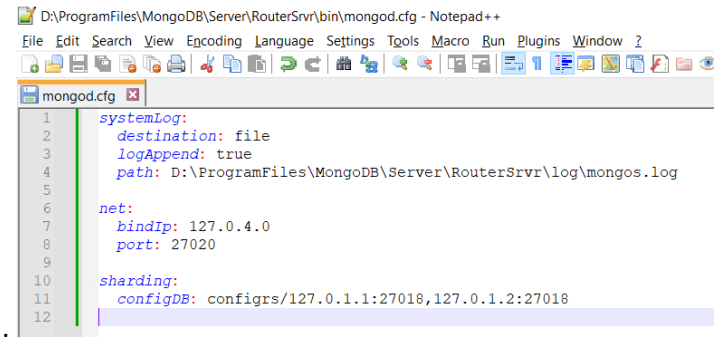
10.5. Start Query Router:

Now, we will setup a query router (mongos) instance such that it is running at IP address `127.0.4.0` on port `27020`.

10.5.1. Configure Router Instance:

Go to `D:\ProgramFiles\MongoDB\Server\RouterSrvr\bin` and create `mongos.cfg` file and add the below contents into it.

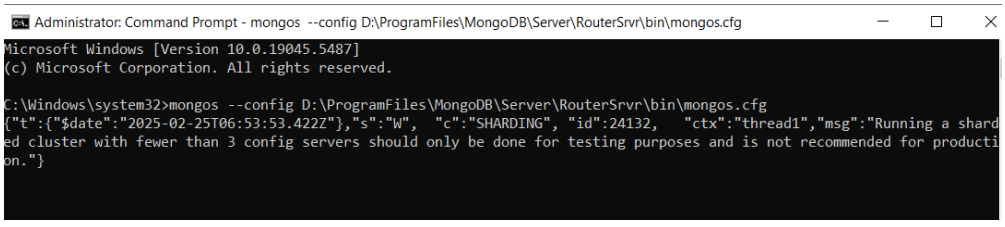
```
systemLog:
  destination: file
  logAppend: true
  path: D:\ProgramFiles\MongoDB\Server\RouterSrvr\log\mongos.log

net:
  bindIp: 127.0.4.0
  port: 27020

sharding:
  configDB: configrs/127.0.1.1:27018,127.0.1.2:27018
```



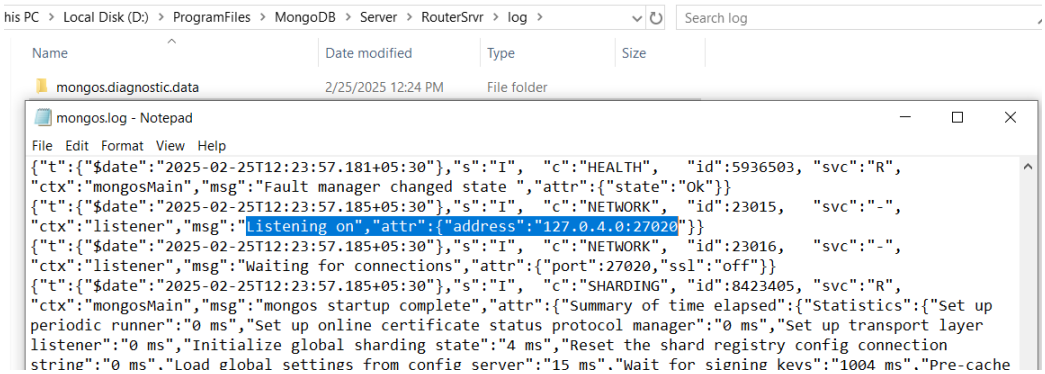
10.5.2. Start Router Instance:

Open a new **Command Prompt** application as **Administrator** and execute the below command to start `RouterSrvr` instance.

```
mongos --config
D:\ProgramFiles\MongoDB\Server\RouterSrvr\bin\mongos.cfg
```

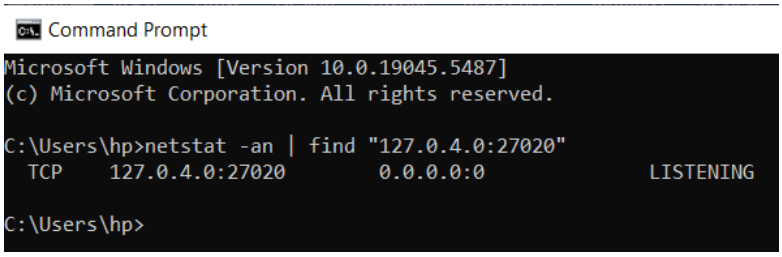


Do not close this window. In couple of minutes, MongoDB instance would be started. You can open `mongod.log` file from `D:\ProgramFiles\MongoDB\Server\RouterSrvr\log` and look for messages **"Listening on address 127.0.4.0:27020"** which indicate that MongoDB instance has been started successfully.



You can also verify if the address `127.0.4.0:27020` is listening by running the following command on another **Command Prompt** window:

```
netstat -an | find "127.0.4.0:27020"
```

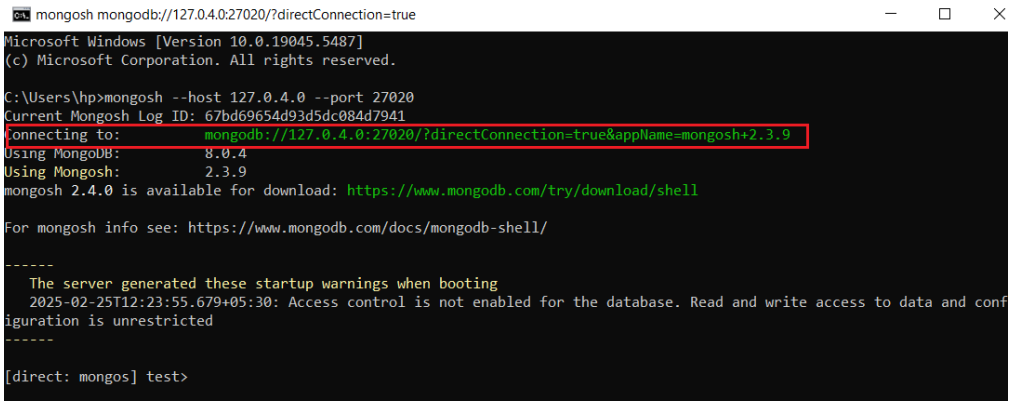


10.6. Add Shards to Cluster:

Now, it's time to add the shards (shard replica sets) to the cluster.

Open a new **Command Prompt** and execute the below command to connect to the query router (mongos) instance listening at `127.0.4.0:27020` address.

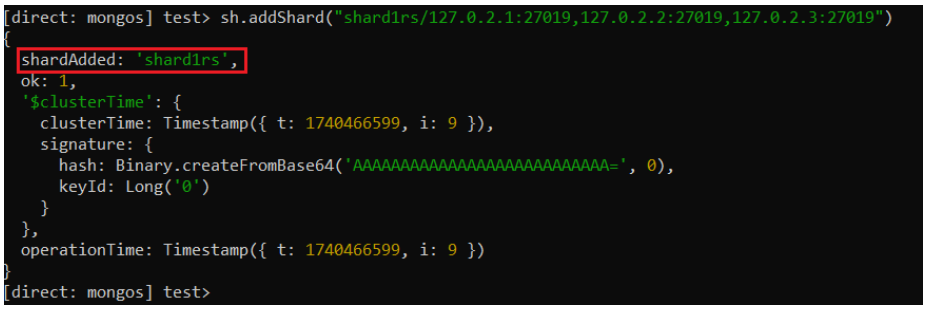
```
mongosh --host 127.0.4.0 --port 27020
```



When you are connected successfully, it will give you `[direct: mongos]test>` prompt to access the `test` database by default in `mongos`.

On the `test>` prompt, run the following command to add the first shard replica to the cluster

```
sh.addShard("shard1rs/127.0.2.1:27019,127.0.2.2:27019,127.0.2.3:27019
")
```



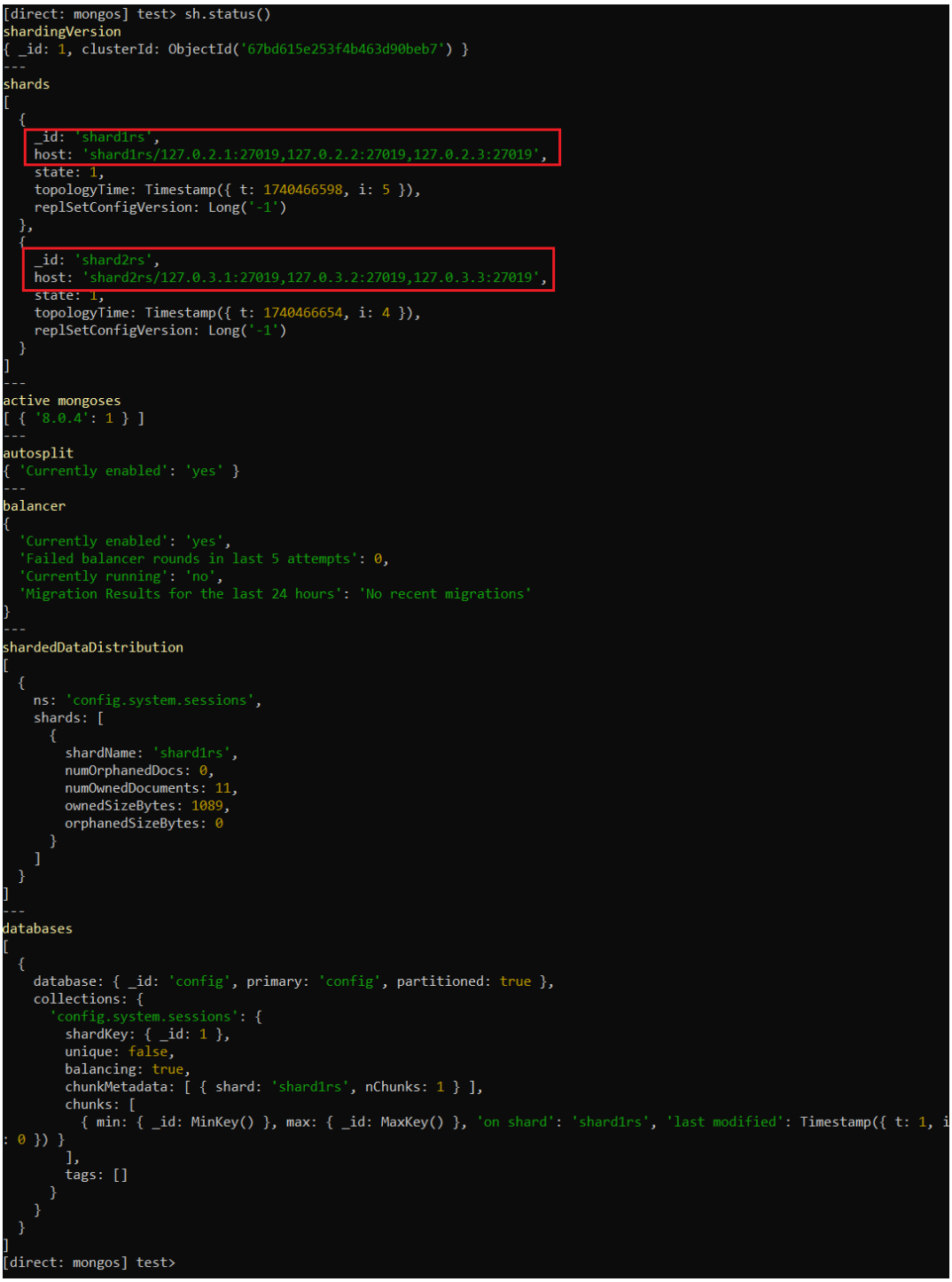
If the shard addition is successful, the `sh.addShard()` method returns the shard name that was added.

Next, run the following command to add the second shard replica to the cluster

```
sh.addShard("shard2rs/127.0.3.1:27019,127.0.3.2:27019,127.0.3.3:27019
")
```

Let's check if shards have been added properly using the following command:

```
sh.status()
```



As you can see above, two shards `shard1rs` and `shard2rs` got added where each shard contains 3 hosts.

10.7. Enable Sharding:

To take the advantage of sharded cluster, we must enable sharding for the database and collection to store data across multiple shards within cluster.

10.7.1. Enable Sharding for Database:

After the sharded cluster is setup, it is important to enable sharding for the database within cluster to store data across shards. Otherwise, the data being inserted into MongoDB cluster is non-partitioned and stored in the primary shard only.

To move further, we will work on the sample sales data which will be stored in a collection called `sales` and a database called `superstore`. The `sales` collection will have documents with the following fields.

```
{
    "category": "Furniture",
    "subcategory": "Bookcases",
    "productname": "Bookcase",
    "city": "Henderson",
    "state": "Kentucky",
    "country": "United States",
    "region": "South",
    "quantity": 2,
    "sales": 261.96
  }
```

On the `mongos` command prompt, run the following command to enable sharding for `superstore` database even if this database doesn't exist yet.

```
sh.enableSharding("superstore")
```

```
[direct: mongos] test> sh.enableSharding("superstore")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1740491426, i: 7 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1740491426, i: 4 })
}
[direct: mongos] test>
```

10.7.2. Enable Sharding for Collection:

Now that the database is configured to allow partitioning, let us enable partitioning or sharding for the `sales` collection even if this collection does not exist yet.

MongoDB provides two ways to shard collections:

- **Range-based sharding** which allows sharding data based on ranges of a given value. It can use multiple fields as the shard key and divides data into contiguous ranges determined by the shard key values.
- **Hashed sharding** which uses a hashed index of a single field as the shard key to partition data across the sharded cluster.

It is always recommended to choose the field as sharding key which has high cardinality i.e. highest number of duplicate values.

Run the following command to enable hased sharding for `sales` collection with `category` field as shard key. In the below command, `category` field is set to hashed which indicates that hashed sharding is enabled on `category` *(if you want to enable ranged-based sharding, then set the `category` field to `1`)*

```
sh.shardCollection("superstore.sales", { "category": "hashed" })
```

```
[direct: mongos] superstore> sh.shardCollection("superstore.sales", { "category": "hashed" })
{
  collectionsharded: 'superstore.sales',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1740498705, i: 5 }),
    signature: {
      hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
      keyId: Long('0')
    }
  },
  operationTime: Timestamp({ t: 1740498705, i: 5 })
}
[direct: mongos] superstore>
```

10.7.3. Insert Documents into Collection:

Now, let us insert some sample documents into `sales` sharded collection.

First, switch to `superstore` database using the below command:

```
use superstore
```

```
[direct: mongos] test> use superstore
switched to db superstore
[direct: mongos] superstore>
```

Then, run the following command to insert 25 documents into `sales` collection.

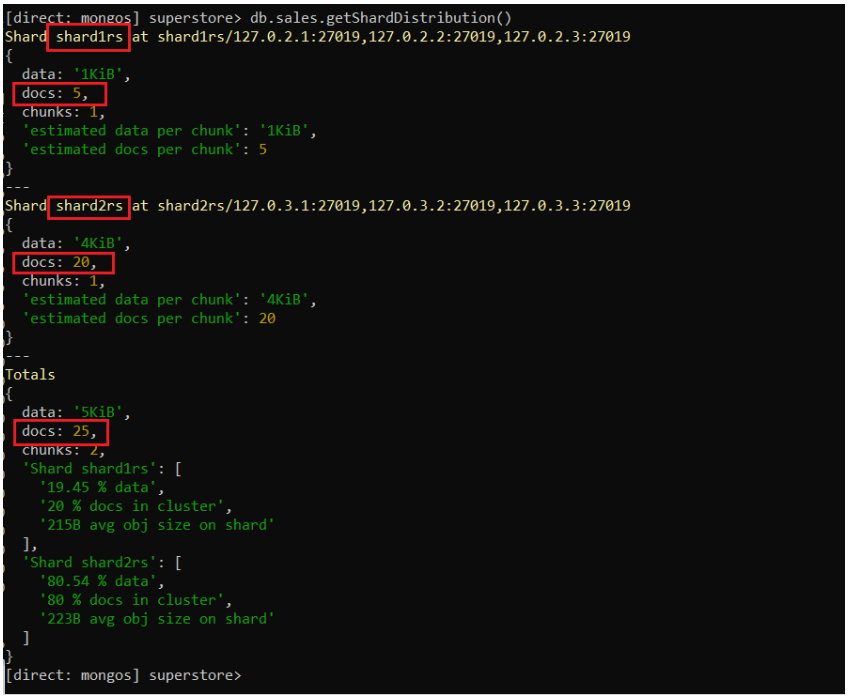
```
db.sales.insertMany([
  {"category": "Furniture", "subcategory": "Bookcases", "productname": "Bookcase",
"city": "Henderson", "state": "Kentucky", "country": "United States", "region":
"South", "quantity": 2, "sales": 261.96},
  {"category": "Office Supplies", "subcategory": "Labels", "productname": "Address
Labels for Typewriters", "city": "Los Angeles", "state": "California", "country":
"United States", "region": "West", "quantity": 2, "sales": 14.62},
  {"category": "Furniture", "subcategory": "Tables", "productname": "CR4500 Table",
"city": "Fort Lauderdale", "state": "Florida", "country": "United States",
"region": "South", "quantity": 5, "sales": 957.5775},
  {"category": "Furniture", "subcategory": "Furnishings", "productname": "Cherry
Wood", "city": "Los Angeles", "state": "California", "country": "United States",
"region": "West", "quantity": 7, "sales": 48.86},
  {"category": "Office Supplies", "subcategory": "Paper", "productname": "Xerox
1967", "city": "Concord", "state": "North Carolina", "country": "United States",
"region": "South", "quantity": 3, "sales": 15.552},
  {"category": "Office Supplies", "subcategory": "Binders", "productname": "Comb
Binding Machine", "city": "Seattle", "state": "Washington", "country": "United
States", "region": "West", "quantity": 3, "sales": 407.976},
  {"category": "Office Supplies", "subcategory": "Appliances", "productname": "HEPA
Air Cleaner", "city": "Fort Worth", "state": "Texas", "country": "United States",
"region": "Central", "quantity": 5, "sales": 68.81},
  {"category": "Office Supplies", "subcategory": "Storage", "productname": "Stur-D-
Stor Shelf", "city": "Madison", "state": "Wisconsin", "country": "United States",
"region": "Central", "quantity": 6, "sales": 665.88},
  {"category": "Office Supplies", "subcategory": "Storage", "productname": "Super
Drawer", "city": "West Jordan", "state": "Utah", "country": "United States",
"region": "West", "quantity": 2, "sales": 55.5},
  {"category": "Office Supplies", "subcategory": "Art", "productname": "Newell
318", "city": "Fremont", "state": "Nebraska", "country": "United States", "region":
"Central", "quantity": 7, "sales": 19.46},
  {"category": "Furniture", "subcategory": "Chairs", "productname": "Stacking
Chair", "city": "Philadelphia", "state": "Pennsylvania", "country": "United
States", "region": "East", "quantity": 2, "sales": 71.372},
  {"category": "Furniture", "subcategory": "Tables", "productname": "CR4500 Table",
"city": "Orem", "state": "Utah", "country": "United States", "region": "West",
"quantity": 3, "sales": 1044.63},
  {"category": "Office Supplies", "subcategory": "Binders", "productname":
"Binders", "city": "Los Angeles", "state": "California", "country": "United
States", "region": "West", "quantity": 2, "sales": 11.648},
  {"category": "Office Supplies", "subcategory": "Paper", "productname": "Easy-
staple paper", "city": "Houston", "state": "Texas", "country": "United States",
"region": "Central", "quantity": 3, "sales": 29.472},
  {"category": "Technology", "subcategory": "Phones", "productname": "GE 30524EE4",
"city": "Richardson", "state": "Texas", "country": "United States", "region":
"Central", "quantity": 7, "sales": 1097.544},
  {"category": "Office Supplies", "subcategory": "Envelopes", "productname":
"Diagonal Seam Envelopes", "city": "Houston", "state": "Texas", "country": "United
States", "region": "Central", "quantity": 9, "sales": 113.328},
  {"category": "Technology", "subcategory": "Phones", "productname": "Panasonic Kx-
TS550", "city": "Naperville", "state": "Illinois", "country": "United States",
"region": "Central", "quantity": 4, "sales": 147.168},
```

```
  {"category": "Office Supplies", "subcategory": "Storage", "productname":
"stackable storage shelf", "city": "Los Angeles", "state": "California", "country":
"United States", "region": "West", "quantity": 2, "sales": 77.88},
  {"category": "Office Supplies", "subcategory": "Storage", "productname": "Smoke
Drawers", "city": "Melbourne", "state": "Florida", "country": "United States",
"region": "South", "quantity": 2, "sales": 95.616},
  {"category": "Office Supplies", "subcategory": "Binders", "productname":
"Binders", "city": "Eagan", "state": "Minnesota", "country": "United States",
"region": "Central", "quantity": 2, "sales": 17.46},
  {"category": "Office Supplies", "subcategory": "Storage", "productname": "Panel
Bin", "city": "Westland", "state": "Michigan", "country": "United States",
"region": "Central", "quantity": 4, "sales": 211.96},
  {"category": "Office Supplies", "subcategory": "Art", "productname": "Chalk
Sticks", "city": "Troy", "state": "New York", "country": "United States", "region":
"East", "quantity": 1, "sales": 1.68},
  {"category": "Technology", "subcategory": "Accessories", "productname":
"Recordable Disc", "city": "Los Angeles", "state": "California", "country": "United
States", "region": "West", "quantity": 2, "sales": 13.98},
  {"category": "Office Supplies", "subcategory": "Binders", "productname": "Index
Sets", "city": "New York City", "state": "New York", "country": "United States",
"region": "East", "quantity": 1, "sales": 4.616},
  {"category": "Office Supplies", "subcategory": "Paper", "productname": "Telephone
Message Books", "city": "Jackson", "state": "Michigan", "country": "United States",
"region": "Central", "quantity": 3, "sales": 19.05}
])
```

When the above documents are inserted, MongoDB will automatically distribute documents across shards based on the values of the `category` field that is defined as shard key.

Use the following command to see how documents were distributed across shards

```
db.sales.getShardDistribution()
```



As you can see above that the collection consists of total `25` documents out of which `5` documents were distributed on the first shard `shard1rs` and `20` documents were distributed on the second shard `shard2rs`.

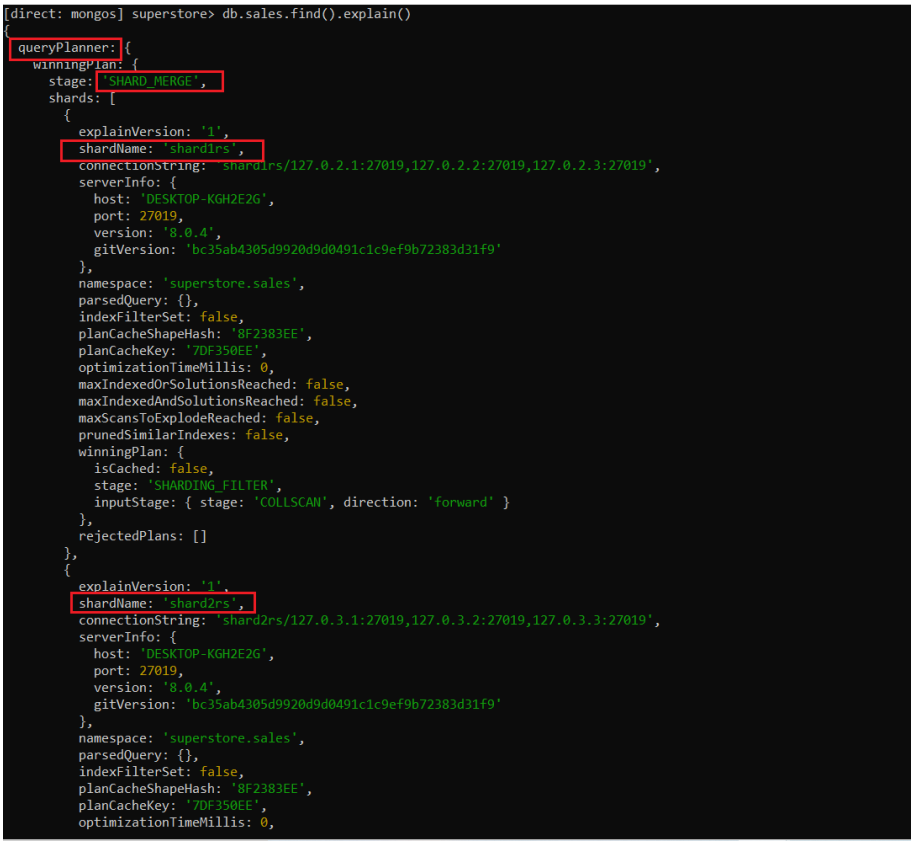
10.7.4. Analyze Shard Usage:

Sharding helps to improve the performance of the system. Based on the query executed, it scans either a single shard or all shards in the entire cluster.

By using the `explain` feature on the query cursor, we can check whether the query spans one or multiple shards. In turn, it can also help to determine whether the query will overload the cluster by reaching out to every shard at once.

Run the following command to see how the query has been executed

```
db.sales.find().explain()
```

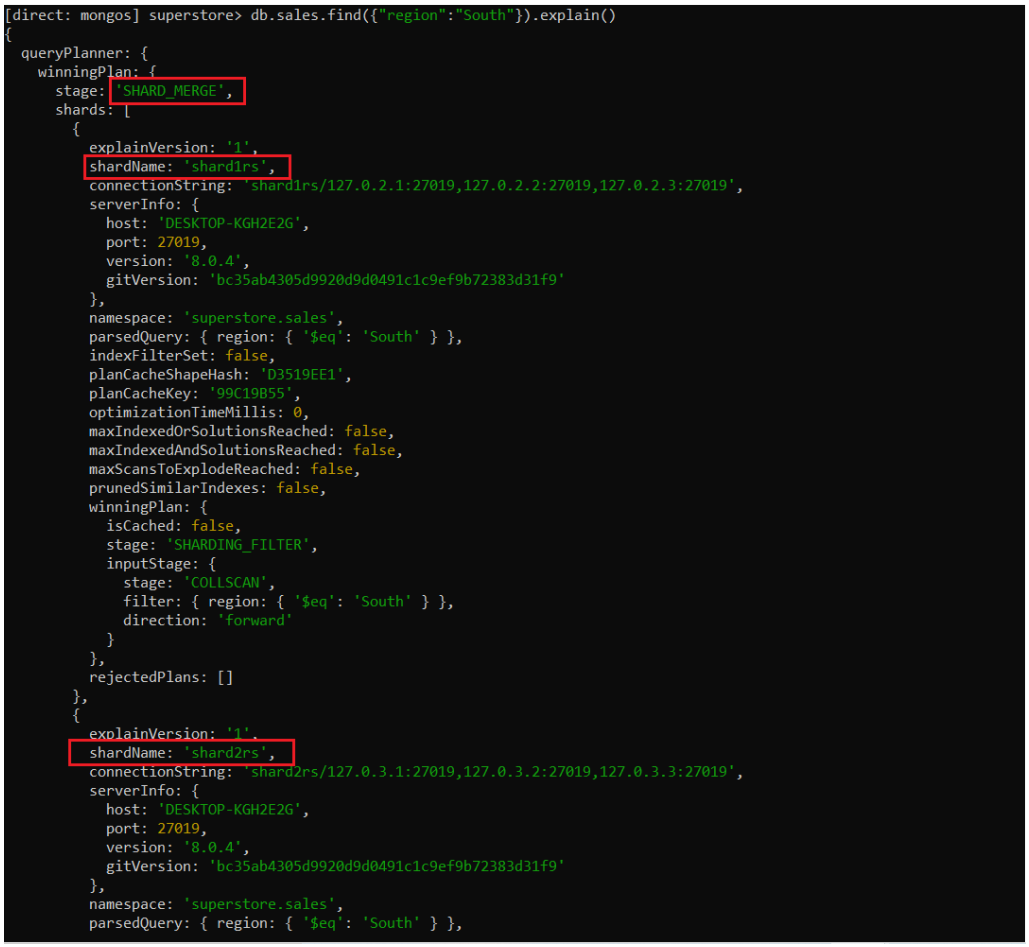


```
[direct: mongos] superstore> db.sales.find().explain()
{
  queryPlanner: {
    winningPlan: {
      stage: 'SHARD_MERGE',
      shards: [
        {
          explainVersion: '1',
          shardName: 'shard1rs',
          connectionString: 'shard1rs/127.0.2.1:27019,127.0.2.2:27019,127.0.2.3:27019',
          serverInfo: {
            host: 'DESKTOP-KGH2E2G',
            port: 27019,
            version: '8.0.4',
            gitVersion: 'bc35ab4305d9920d9d0491c1c9ef9b72383d31f9'
          },
          namespace: 'superstore.sales',
          parsedQuery: {},
          indexFilterSet: false,
          planCacheShapeHash: '8F2383EE',
          planCacheKey: '7DF350EE',
          optimizationTimeMillis: 0,
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          prunedSimilarIndexes: false,
          winningPlan: {
            isCached: false,
            stage: 'SHARDING_FILTER',
            inputStage: { stage: 'COLLSCAN', direction: 'forward' }
          },
          rejectedPlans: []
        },
        {
          explainVersion: '1',
          shardName: 'shard2rs',
          connectionString: 'shard2rs/127.0.3.1:27019,127.0.3.2:27019,127.0.3.3:27019',
          serverInfo: {
            host: 'DESKTOP-KGH2E2G',
            port: 27019,
            version: '8.0.4',
            gitVersion: 'bc35ab4305d9920d9d0491c1c9ef9b72383d31f9'
          },
          namespace: 'superstore.sales',
          parsedQuery: {},
          indexFilterSet: false,
          planCacheShapeHash: '8F2383EE',
          planCacheKey: '7DF350EE',
          optimizationTimeMillis: 0,
          maxIndexedOrSolutionsReached: false,
          maxIndexedAndSolutionsReached: false,
          maxScansToExplodeReached: false,
          prunedSimilarIndexes: false,
          winningPlan: {
            isCached: false,
            stage: 'SHARDING_FILTER',
            inputStage: { stage: 'COLLSCAN', direction: 'forward' }
          },
          rejectedPlans: []
        }
      ]
    }
  },
  queryShapeHash: '2ED38F9C90C3912E7ED06275EA2F1D401F7F6DC898AFAE1DA2B4DC5B5C10E865',
  serverInfo: {
    host: 'DESKTOP-KGH2E2G',
    port: 27020,
    version: '8.0.4',
    gitVersion: 'bc35ab4305d9920d9d0491c1c9ef9b72383d31f9'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes: 104857600,
    internalQueryFacetMaxOutputDocSizeBytes: 104857600,
    internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,
    internalDocumentSourceGroupMaxMemoryBytes: 104857600,
    internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,
    internalQueryProhibitBlockingMergeOnMongoS: 0,
    internalQueryMaxAddToSetBytes: 104857600,
    internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,
    internalQueryFrameworkControl: 'trySbeRestricted',
    internalQueryPlannerIgnoreIndexWithCollationForRegex: 1
  },
  command: {
    find: 'sales',
    filter: {},
    lsid: { id: UUID('b866c2b6-3575-4ee1-9785-03722b380987') },
    '$clusterTime': {
      clusterTime: Timestamp({ t: 1740501666, i: 2 }),
      signature: {
        hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
        keyId: 0
      }
    },
    '$db': 'superstore'
  },
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1740501860, i: 1 }),
```

As you can see above, the query planner used **SHARD_MERGE** strategy, nothing but multiple shards were used to resolve the query and provides us the list of shards taking part in the evaluation.

Now, let's run the following command to filter sales data that belongs `South region` and see how the query is executed.

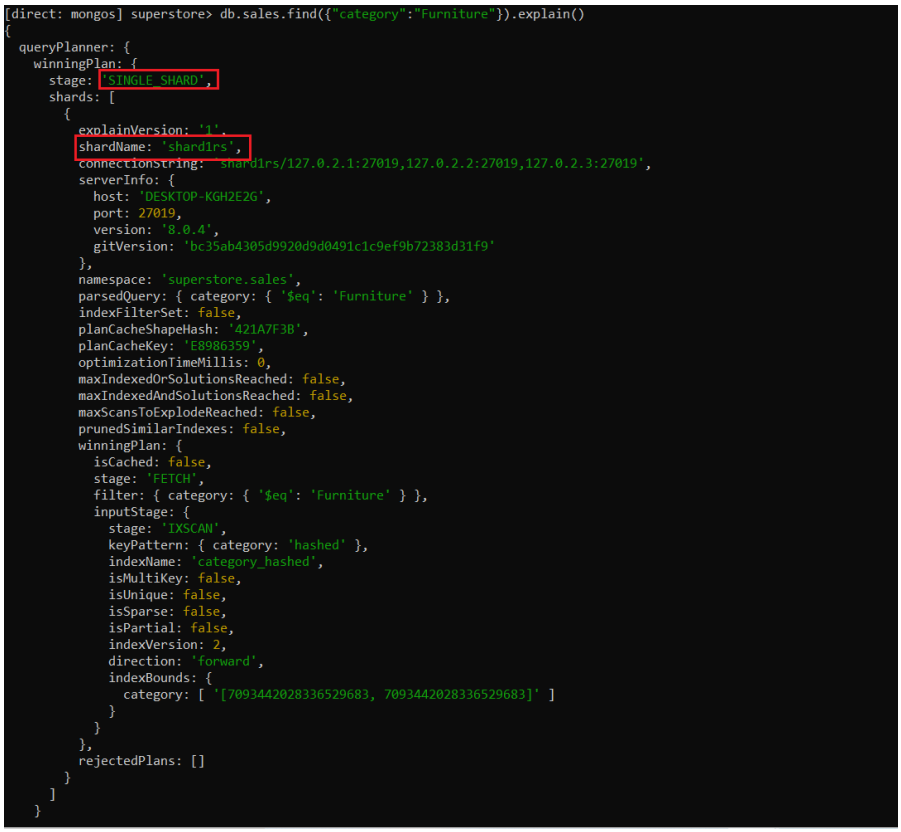
```
db.sales.find({"region":"South"}).explain()
```



This time also, the query planner has used **SHARD_MERGE** strategy to retrieve documents.

When you filter against the shard key, query planner should a different strategy.
Run the following command to see the query planner when filtering `sales` data with `category=Furniture`. Remember that we selected `category` field as shard key while sharding `sales` collection.

```
db.sales.find({"region":"South"}).explain()
```



As you see above, this time, MongoDB has used a different query strategy `SINGLE_SHARD` instead of `SHARD_MERGE`. This means only a single shard was needed to satisfy the query. In this example, documents for `Furniture` `category` were stored on the first shard in the cluster.

11. PyMongo:

While MongoDB can be accessed from various programming languages including C, C+, Java, Python etc., it is easy to use **Python** language to query MongoDB.

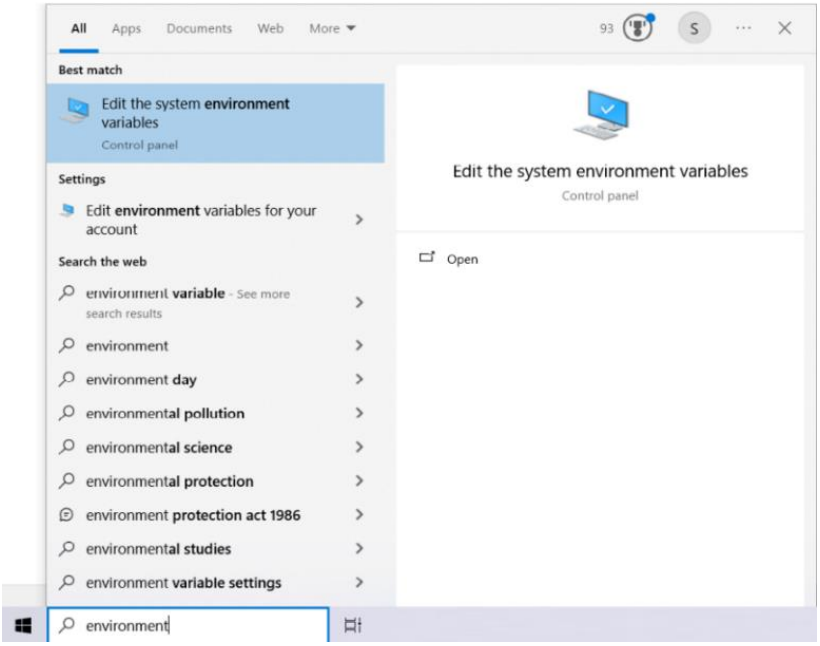
PyMongo is a library available in the Python language to work with MongoDB. In other words, Python needs a MongoDB driver called PyMongo to access the MongoDB database.

11.1. Install Python:

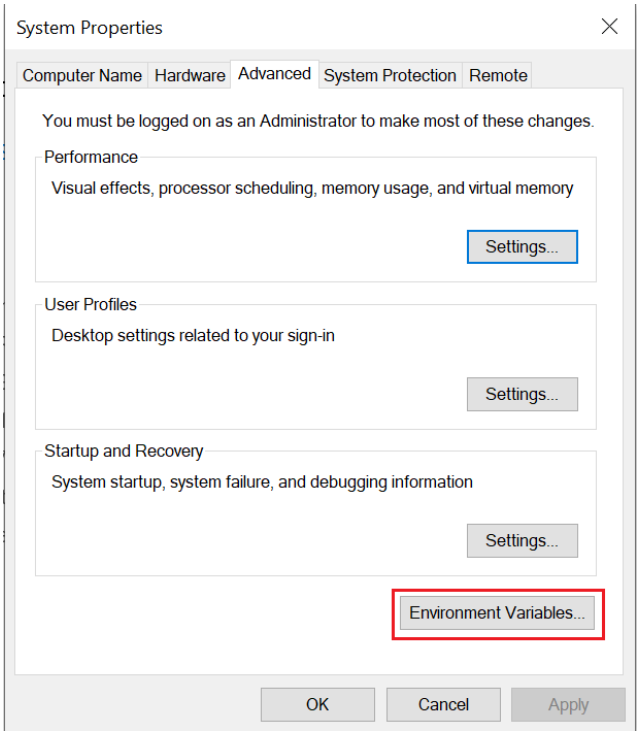
If you do not have Python installed already in your system, you can download and install the latest version from the official Python Downloads website.

After you installed python, you need to configure the `PATH` environment variable defining the Python installation path.

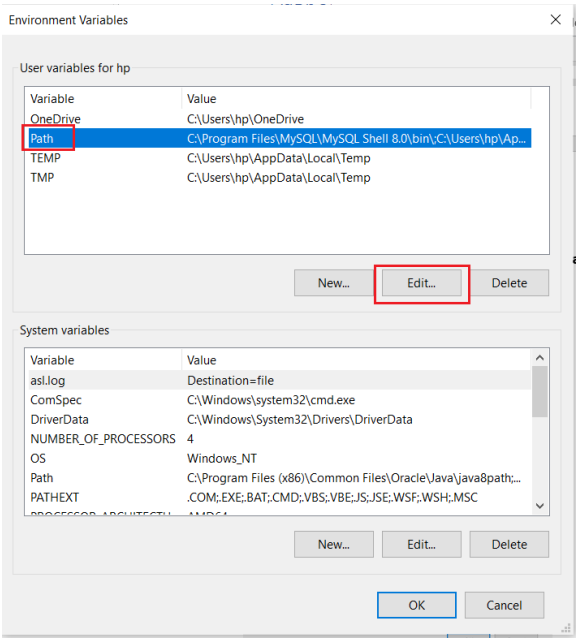
In the Windows search bar, start typing “environment variables” and select the first match which opens up **System Properties** dialog.



On the **System Properties** window, press **Environment Variables** button.



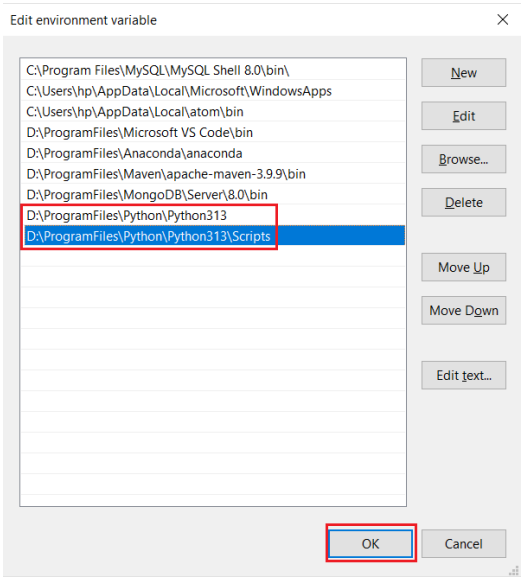
In the **Environment Variables** dialog, select `PATH` variable under **User Variables** and press **Edit** button.



Press **New** and add the location of you python installation and its `scripts` directory. Then press OK *(In my case, python was installed under* `D:\ProgramFiles\Python\Python313` *and so, added the below values).*

`D:\ProgramFiles\Python\Python313`

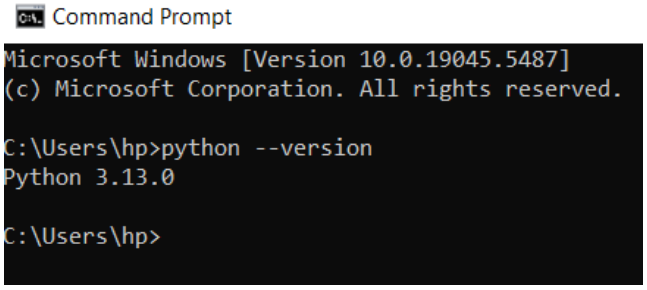
`D:\ProgramFiles\Python\Python313\Scripts`



Then, press **OK** again to apply environment variable changes and close window.

Open a new Command Prompt, and run the following command to verify the python version.

```
python --version
```



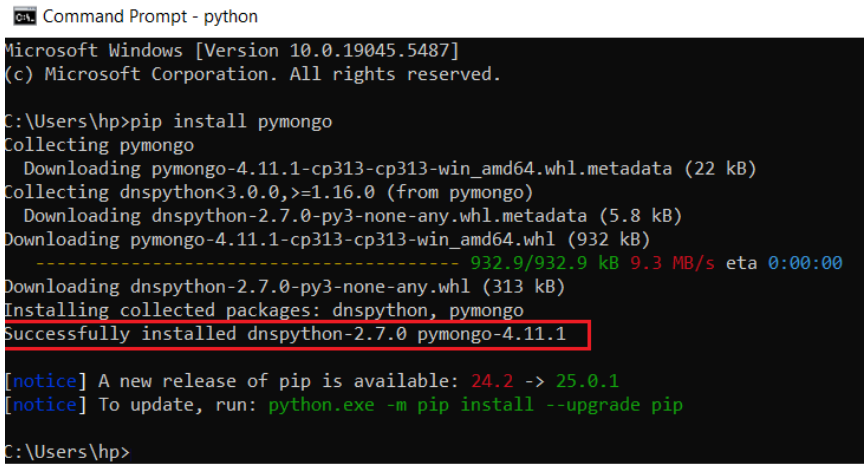
As you see above, it is showing that **Python 3.13.0** version was installed.

11.2. Install PyMongo:

Once you have Python installed in your system, you need to install the `pymongo` library into Python.

Open a new **Command Prompt** and run the below command to install `pymongo` library.

```
pip install pymongo
```

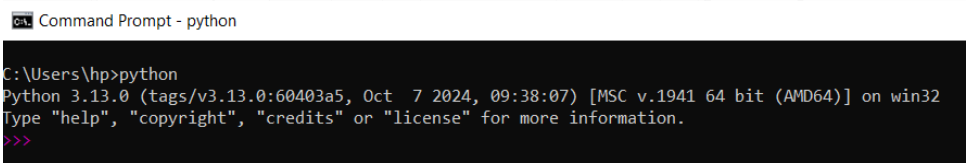


11.3. Connect MongoDB:

Now let's try to connect our MongoDB sharded cluster that we deployed earlier from Python.

First, start the `python` shell using the below command.

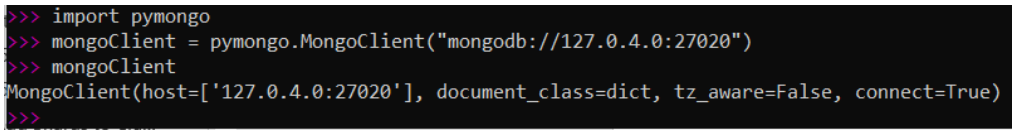
```
python
```



When python shell is started, it gives you >>> prompt to execute Python commands.

On the >>> prompt, run the following commands to import `pymongo` library and create a `MongoClient` object using the connection URL `mongodb://127.0.4.0:27020`. Here, we are connecting to mongos (query router) host to access MongoDB.

```
import pymongo
mongoClient = pymongo.MongoClient("mongodb://127.0.4.0:27020")
mongoClient
```

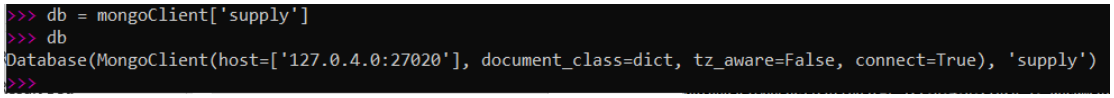


11.4. Create Database:

Once the `MongoClient` object is created successfully, we can access see the available databases and connect to a specific database.

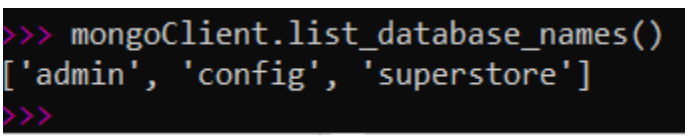
Let us create a `db` object to access the `supply` database *(this database does not exist yet)*. MongoDB will not create a database until a collection is created with at least one document.

```
db = mongoClient['supply']
db
```



Use the below command to list down the available databases in the Mongo cluster:

```
mongoClient.list_database_names()
```



As you see above, the `supply` database is not yet created.

11.5. Create Collection:

Once the `db` object is created successfully, we can access see the available collections and connect to a specific collection.

Let us create a `col` object to access the `suppliers` collection (this collection is not yet available). Use the below command to create a collection in MongoDB by specifying the collection name in [] brackets. MongoDB will not create a collection until at least one document is loaded into it.

```
col = db['suppliers']
col
```

```
>>> col = db['suppliers']
>>> col
Collection(Database(MongoClient(host=['127.0.4.0:27020'], document_class=dict, tz_aware=False, connect=True), 'supply'),
 'suppliers')
>>>
```

In the above command, we can call the collection name in two ways such as `db['suppliers']` or `db.suppliers`

Use the below command to list down the available collections in the database:

```
db.list_collection_names()
```

```
>>> db.list_collection_names()
[]
>>>
```

As you see above, the `suppliers` collection is not yet created.

11.6. Insert Documents:

PyMongo collection object has `insert_one()` or `insert_many()` methods to insert a single or multiple documents. Document data must be provided in JSON format with key-value pairs.

Use `insert_one()` method to insert one document into the collection.
In the below command, `col` is the collection object that we created earlier and is pointed to `suppliers`. MongoDB assigns a unique id (`ObjectId`) for each document inserted when the document data does not contain `_id` field.

```
col.insert_one({"name": "Exotic Liquids", "address": "49 Gilbert
St.", "city": "London", "country": "UK"})
```

```
>>> col.insert_one({"name": "Exotic Liquids", "address": "49 Gilbert St.", "city": "London", "country": "UK"})
InsertOneResult(ObjectId('67c55e5d83f28d1723adeffa'), acknowledged=True)
>>>
```

The `insert_one()` method returns the `InsertOneResult` object which contains the `ObjectId` of the inserted document.

You can print the `ObjectId` of the inserted document using `inserted_id` property of `InsertOneResult` object by storing the result of `insert_one()` method into a variable as below:

```
ins_res = col.insert_one({"name": "Tokyo Traders", "address": "9-8
Sekimai Musashino-shi", "city": "Tokyo", "country": "Japan"})
print(ins_res.inserted_id)
```

```
>>> ins_res = col.insert_one({"name": "Tokyo Traders", "address": "9-8 Sekimai Musashino-shi", "city": "Tokyo", "countr\
y": "Japan"})
>>> print(ins_res.inserted_id)
67c5686783f28d1723adf001
>>>
```

Now, use the `list_collection_names()` method to see if the `suppliers` collection is created:

```
db.list_collection_names()
```

```
>>> db.list_collection_names()
['suppliers']
>>>
```

Use `insert_many()` method to insert multiple documents at once into the collection. In this method, all documents must be specified in list form. The `insert_many()` method returns the `InsertManyResult` object that contains `ObjectId` of all inserted documents which you can print using `inserted_ids` property of `InsertManyResult` object by storing the result of `insert_many()` method into a variable as below:

```
ins_many_res = db.suppliers.insert_many([
{"name": "Ma Maison", "address": "2960 Rue St. Laurent", "city":
"Montreal", "country": "Canada"},
{"name": "Bigfoot Breweries", "address": "3400 - 8th Avenue Suite
210", "city": "Bend", "country": "USA"},
{"name": "Leka Trading", "address": "471 Serangoon Loop", "city": "",
"country": "Singapore"},
{"name": "Specialty Biscuits", "address": "29 King's Way", "city":
"Manchester", "country": "USA"}
])
```

```
print(ins_many_res.inserted_ids)
```

```
>>> ins_many_res = db.suppliers.insert_many([
... {"name": "Ma Maison", "address": "2960 Rue St. Laurent", "city": "Montreal", "country": "Canada"},
... {"name": "Bigfoot Breweries", "address": "3400 - 8th Avenue Suite 210", "city": "Bend", "country": "USA"},
... {"name": "Leka Trading", "address": "471 Serangoon Loop", "city": "", "country": "Singapore"},
... {"name": "Specialty Biscuits", "address": "29 King's Way", "city": "Manchester", "country": "USA"}
... ])
>>>
>>> print(ins_many_res.inserted_ids)
[ObjectId('67c568a583f28d1723adf002'), ObjectId('67c568a583f28d1723adf003'), ObjectId('67c568a583f28d1723adf004'), Objec
tId('67c568a583f28d1723adf005')]
>>>
```

11.7. Find Documents:

PyMongo collection object has `find_one()` and `find()` methods to retrieve a single or multiple documents from a collection. Both these methods can take 2 parameters where the first parameter indicates the JSON form of filter criteria and second parameter indicates the JSON form of fields to be included or excluded.

Use `find_one()` method to select a single document data from the collection. It always returns the first occurrence in the selection.

Run the below command to get the first document in the `suppliers` collection:

```
db.suppliers.find_one()
```

or

```
db.suppliers.find_one({})
```

```
>>> db.suppliers.find_one()
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
>>>
>>> db.suppliers.find_one({})
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
>>>
```

Run the below command to retrieve the first document which contains `country=USA` from the `suppliers` collection:

```
db.suppliers.find_one({"country": "USA"})
```

```
>>> db.suppliers.find_one({"country": "USA"})
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'USA'}
>>>
```

Use `find()` method to retrieve multiple documents from the collection. It returns a cursor object which you must loop in to see the actual document data. The `find()` method without any parameter is equivalent to `SELECT *` command in relational databases.

Run the below command to get all documents from the `suppliers` collection:

```
find_res = db.suppliers.find({})
print(find_res)
for doc in find_res:
	print(doc)
```

```
>>> find_res = db.suppliers.find({})
>>> print(find_res)
<pymongo.synchronous.cursor.Cursor object at 0x0000029FEC70A990>
>>> for doc in find_res:
...     print(doc)
...
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'USA'}
{'_id': ObjectId('67c568a583f28d1723adf004'), 'name': 'Leka Trading', 'address': '471 Serangoon Loop', 'city': '', 'coun
try': 'Singapore'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'USA'}
>>>
```

Run the below command to get all documents which contains `country=USA` from the `suppliers` collection.

```
for doc in db.suppliers.find({"country": "USA"}):
	print(doc)
```

```
>>> for doc in db.suppliers.find({"country": "USA"}):
...     print(doc)
...
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'USA'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'USA'}
>>>
```

Run the below command to get all documents which contains `address` field value starting with `'3'` or greater than `'3'` from the `suppliers` collection:

```
for doc in db.suppliers.find({"address": {"$gt": "3"}}):
	print(doc)
```

```
>>> for doc in db.suppliers.find({"address": {"$gt": "3"}}):
...     print(doc)
...
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'USA'}
{'_id': ObjectId('67c568a583f28d1723adf004'), 'name': 'Leka Trading', 'address': '471 Serangoon Loop', 'city': '', 'coun
try': 'Singapore'}
>>>
```

Run the below command to get all documents which contains `city` starting with `'M'` but this time, exclude *(indicating with `0`)* `_id` field and include *(indicating with `1`)* only `name` and `city` fields in the output:

```
for doc in db.suppliers.find({"city": {"$regex": "^M"}}, {'_id': 0,
"name": 1, "city": 1}):
       print(doc)
```

```
>>> for doc in db.suppliers.find({"city": {"$regex": "^M"}}, {'_id': 0, "name": 1, "city": 1}):
...     print(doc)
...
{'name': 'Ma Maison', 'city': 'Montreal'}
{'name': 'Specialty Biscuits', 'city': 'Manchester'}
>>>
```

Use `limit()` method to limit the result set in MongoDB. This method expects one integer parameter defining the numbers of documents to return.

Run the below command to get top 3 documents from the `suppliers` collection:

```
for doc in db.suppliers.find().limit(3):
       print(doc)
```

```
>>> for doc in db.suppliers.find().limit(3):
...     print(doc)
...
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
>>>
```

Use `sort()` method to sort the result set in either ascending or descending order. This method takes 2 parameters where the first parameter indicates the fieldname and second parameter indicates the sort order *(use `1` for ascending and `-1` descending)*. The default sort order is ascending.

Run the below command to retrieve all documents sorted by `country` in ascending order from `suppliers` collection:

```
for doc in db.suppliers.find().sort("country"):
       print(doc)
```

```
>>> for doc in db.suppliers.find().sort("country"):
...     print(doc)
...
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf004'), 'name': 'Leka Trading', 'address': '471 Serangoon Loop', 'city': '', 'coun
try': 'Singapore'}
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'USA'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'USA'}
>>>
```

Run the below command to retrieve all documents sorted by `country` in descending order from `suppliers` collection:

```
for doc in db.suppliers.find().sort("country", -1):
    print(doc)
```

```
>>> for doc in db.suppliers.find().sort("country", -1):
...     print(doc)
...
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'USA'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'USA'}
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
{'_id': ObjectId('67c568a583f28d1723adf004'), 'name': 'Leka Trading', 'address': '471 Serangoon Loop', 'city': '', 'coun
try': 'Singapore'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
>>>
```

11.8. Count Documents:

Use `count_documents()` method of collection object to get the number of documents available in a collection. This method expects one parameter, just pass the empty dictionary {} to count the total documents in a collection.

```
db.suppliers.count_documents({})
```

```
>>> db.suppliers.count_documents({})
6
>>>
```

You can pass the filter criteria to the `count_documents()` to get the count of documents matching the query criteria.

```
db.suppliers.count_documents({"country": "USA"})
```

```
>>> db.suppliers.count_documents({"country": "USA"})
2
>>>
```

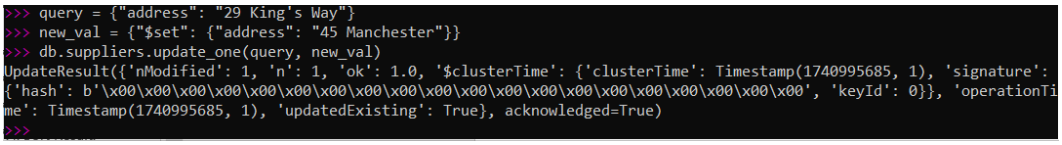
11.9. Update Documents:

PyMongo collection object has `update_one()` and `update_many()` methods to update one or more documents in a collection. Both these methods expect 2 parameters, the first parameter is the query object or filter criteria defining which documents to update and the second parameter is the object defining the new values of the document.

Use `update_one()` method to update a single document data in the collection. If the query to update finds more than one document, then this method updates the first occurrence.

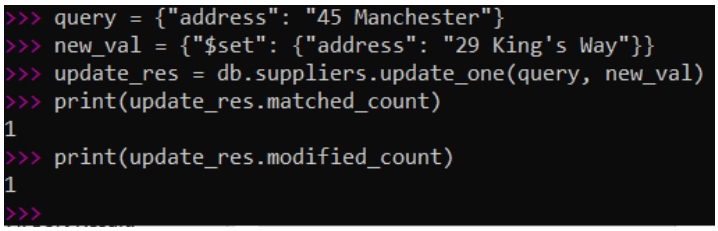
Run the below command to update `address` from `29 King's Way` to `45 Manchester` in the `suppliers` collection:

```
query = {"address": "29 King's Way"}
new_val = {"$set": {"address": "45 Manchester"}}
db.suppliers.update_one(query, new_val)
```



The `update_one()` method returns the `UpdateResult` object which has `matched_count` property defining the number of documents matched with the query specified and `modfied_count` property defining the number of documents updated which you can print as below:

```
query = {"address": "45 Manchester"}
new_val = {"$set": {"address": "29 King's Way"}}
update_res = db.suppliers.update_one(query, new_val)
print(update_res.matched_count)
print(update_res.modified_count)
```



Use `update_many()` method to update multiple documents at once if it meets the query criteria.

Run the below command to update `country` starting with `USA` to `United States` in the `suppliers` collection and print the number of documents that got updated:

```
query = {"country" : {"$regex" : "^US"}}
new_val = {"$set" : {"country" : "United States"}}
update_res = db.suppliers.update_many(query, new_val)
print(f"{update_res.modified_count} documents are updated")
```

```
>>> query = {"country" : {"$regex" : "^US"}}
>>> new_val = {"$set" : {"country" : "United States"}}
>>> update_res = collection.update_many(query, new_val)
>>> print(f"{update_res.modified_count} documents are updated")
2 documents are updated
>>>
```

Then, use the `find()` method to see the latest data:

```
for doc in db.suppliers.find():
       print(doc)
```

```
>>> for doc in db.suppliers.find():
...     print(doc)
...
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids', 'address': '49 Gilbert St.', 'city': 'London', '
country': 'UK'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'United States'}
{'_id': ObjectId('67c568a583f28d1723adf004'), 'name': 'Leka Trading', 'address': '471 Serangoon Loop', 'city': '', 'coun
try': 'Singapore'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'United States'}
>>>
```

PyMongo provides `replace_one()` method to replace an entire document.

Run the below command to replace the first document from the `suppliers` collection where `name : Exotic Liquids` and display the latest data.

```
query = {"name" : "Exotic Liquids"}
new_doc = {"name": "Exotic Liquids Ltd", 'address': '49 Gilbert
Street', 'city': 'London', 'country': 'United Kingdom'}
replace_res = db.suppliers.replace_one(query, new_doc)
print(f"{replace_res.modified_count} documents are replaced")
for doc in db.suppliers.find():
       print(doc)
```

```
>>> query = {"name" : "Exotic Liquids"}
>>> new_doc = {"name": "Exotic Liquids Ltd", 'address': '49 Gilbert Street', 'city': 'London', 'country': 'United Kingd\
om'}
>>> replace_res = db.suppliers.replace_one(query, new_doc)
>>> print(f"{replace_res.modified_count} documents are replaced")
1 documents are replaced
>>> for doc in db.suppliers.find():
...     print(doc)
...
{'_id': ObjectId('67c5685583f28d1723adf000'), 'name': 'Exotic Liquids Ltd', 'address': '49 Gilbert Street', 'city': 'Lon
don', 'country': 'United Kingdom'}
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'United States'}
{'_id': ObjectId('67c568a583f28d1723adf004'), 'name': 'Leka Trading', 'address': '471 Serangoon Loop', 'city': '', 'coun
try': 'Singapore'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'United States'}
>>>
```

11.10. Delete Documents:

PyMongo collection object has `delete_one()` and `delete_many()` methods to delete one or more documents from a collection. Both these methods expects a query or filter criteria defining which documents to be deleted.

Use `delete_one()` method to delete a single document. This method returns the `DeleteResult` object which has `deleted_count` property defining the number of deleted documents which you can print.

Run the below command to delete document where `name: Leka Trading` from the `suppliers` collection and print the number of deleted records:

```
delete_res=db.suppliers.delete_one({"name": "Leka Trading"})
print(delete_res.deleted_count)
```

```
>>> delete_res=db.suppliers.delete_one({"name": "Leka Trading"})
>>> print(delete_res.deleted_count)
1
>>>
```

If you want to delete the first document in the collection, then just pass an empty query using `{}` to `delete_one()` method:

```
db.suppliers.delete_one({})
```

```
>>> db.suppliers.delete_one({})
DeleteResult({'n': 1, 'ok': 1.0, '$clusterTime': {'clusterTime': Timestamp(1740998791, 1), 'signature': {'hash': b'\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00', 'keyId': 0}}, 'operationTime': Timestamp(1
740998791, 1)}, acknowledged=True)
>>>
```

PyMongo also provides `find_one_and_delete()` method to delete a single document based on the filter and sort criteria and returns the deleted document

```
db.suppliers.find_one_and_delete({'name': 'Ma Maison'})
```

```
>>> db.suppliers.find_one_and_delete({'name': 'Ma Maison'})
{'_id': ObjectId('67c568a583f28d1723adf002'), 'name': 'Ma Maison', 'address': '2960 Rue St. Laurent', 'city': 'Montreal'
, 'country': 'Canada'}
>>>
```

Then, use the `find()` method to see the available documents in the collection:

```
for doc in db.suppliers.find():
        print(doc)
```

```
>>> for doc in db.suppliers.find():
...     print(doc)
...
{'_id': ObjectId('67c5686783f28d1723adf001'), 'name': 'Tokyo Traders', 'address': '9-8 Sekimai Musashino-shi', 'city': '
Tokyo', 'country': 'Japan'}
{'_id': ObjectId('67c568a583f28d1723adf003'), 'name': 'Bigfoot Breweries', 'address': '3400 - 8th Avenue Suite 210', 'ci
ty': 'Bend', 'country': 'United States'}
{'_id': ObjectId('67c568a583f28d1723adf005'), 'name': 'Specialty Biscuits', 'address': "29 King's Way", 'city': 'Manches
ter', 'country': 'United States'}
>>>
```

As you see above, out of 6 documents that we inserted earlier, only 3 documents exist currently.

Use `delete_many()` method to delete multiple documents at once if it meets the query criteria.

Run the below command to delete documents where `country : United States` in the `suppliers` collection and print the number of documents that got deleted:

```
delete_res=db.suppliers.delete_many({"country": "United States"})
print(delete_res.deleted_count)
```

```
>>> delete_res=db.suppliers.delete_many({"country": "United States"})
>>> print(delete_res.deleted_count)
2
>>>
```

If you want to delete all documents in the collection, then just pass an empty query using `{}` to `delete_many()` method:

```
del_many_res = db.suppliers.delete_many({})
print(f"{del_many_res.deleted_count} documents were deleted")
print(f"Total documents available:
{db.suppliers.count_documents({})}")
```

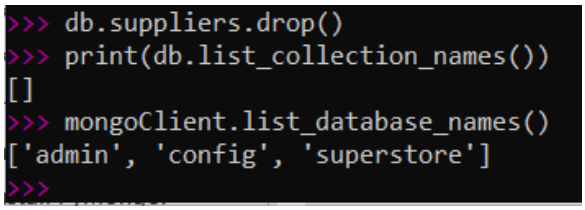
```
>>> del_many_res = db.suppliers.delete_many({})
>>> print(f"{del_many_res.deleted_count} documents were deleted")
1 documents were deleted
>>> print(f"Total documents available: {db.suppliers.count_documents({})}")
Total documents available: 0
>>>
```

11.11. Drop Collection:

PyMongo collection object has `drop()` method to drop the collection from the database. This method returns True if the collection was dropped successfully or False if the collection does not exist. If the database consists of single collection which gets dropped, then database also gets dropped automatically.

Run the below commands to drop `suppliers` collection and see the list of available collections and databases:

```
db.suppliers.drop()
print(db.list_collection_names())
mongoClient.list_database_names()
```



As you see above, as soon as `suppliers` collection was dropped, `supply` database got dropped automatically since it consists of `suppliers` collection only.

Congratulations!! You have successfully installed MongoDB with replica sets and sharding cluster configuration and executed MongoDB queries using Mongo Shell and Mongo Compass utilities. You have also seen how to work with MongoDB using Python scripting language.