Mongo Document operations & Backend Integration

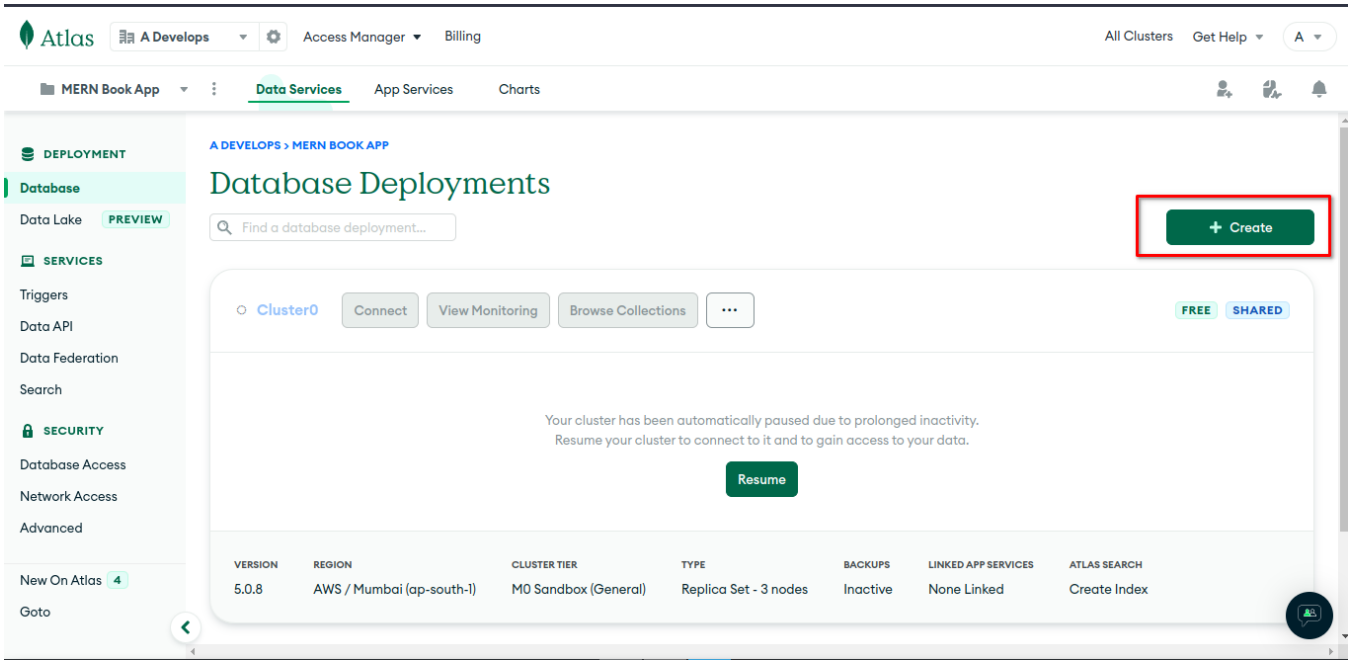
Installing the MongoDB Node.js Driver

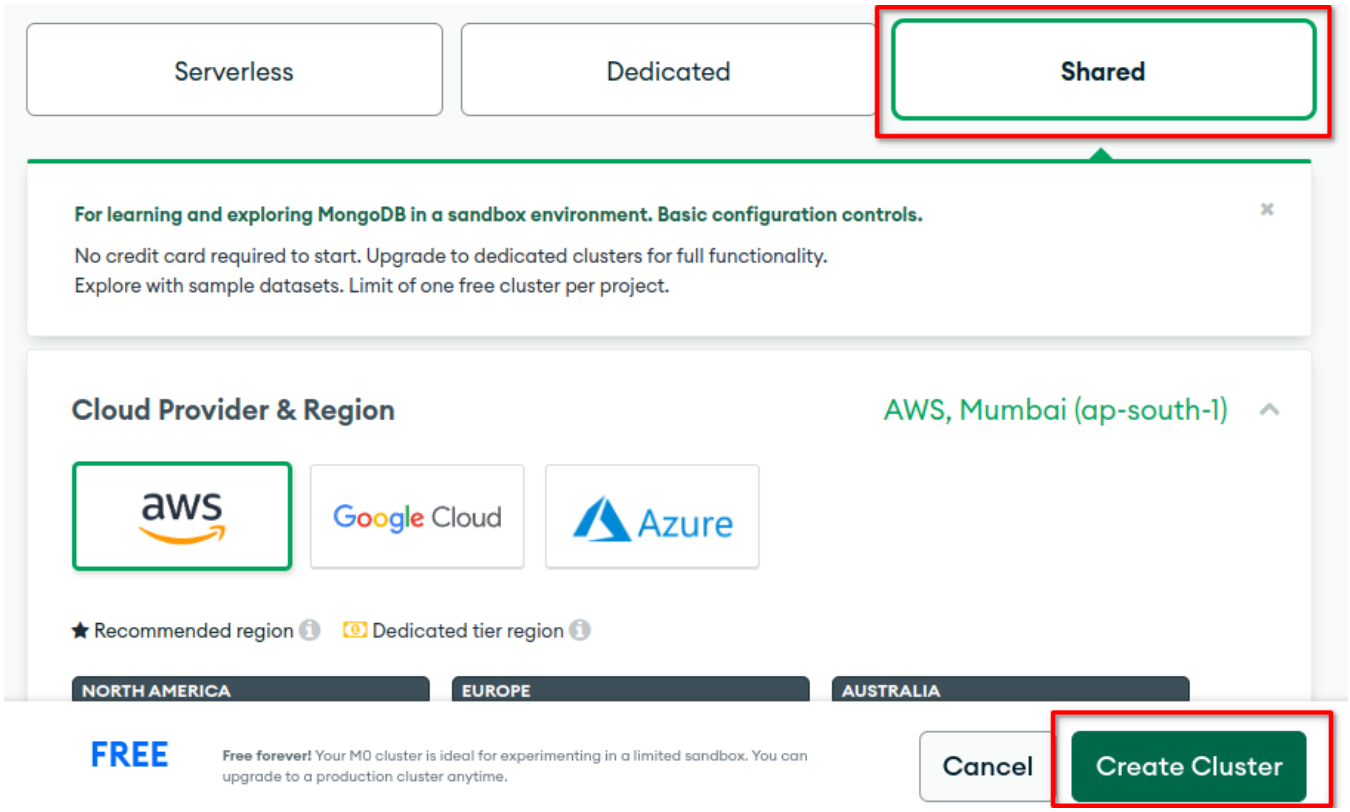
1. First create a new project folder and initialize it with npm by running `npm init -y` command.
2. Install the `mongodb` package by the command `npm i mongodb` .
   It will not reflect in the package.jason under the dependency like :

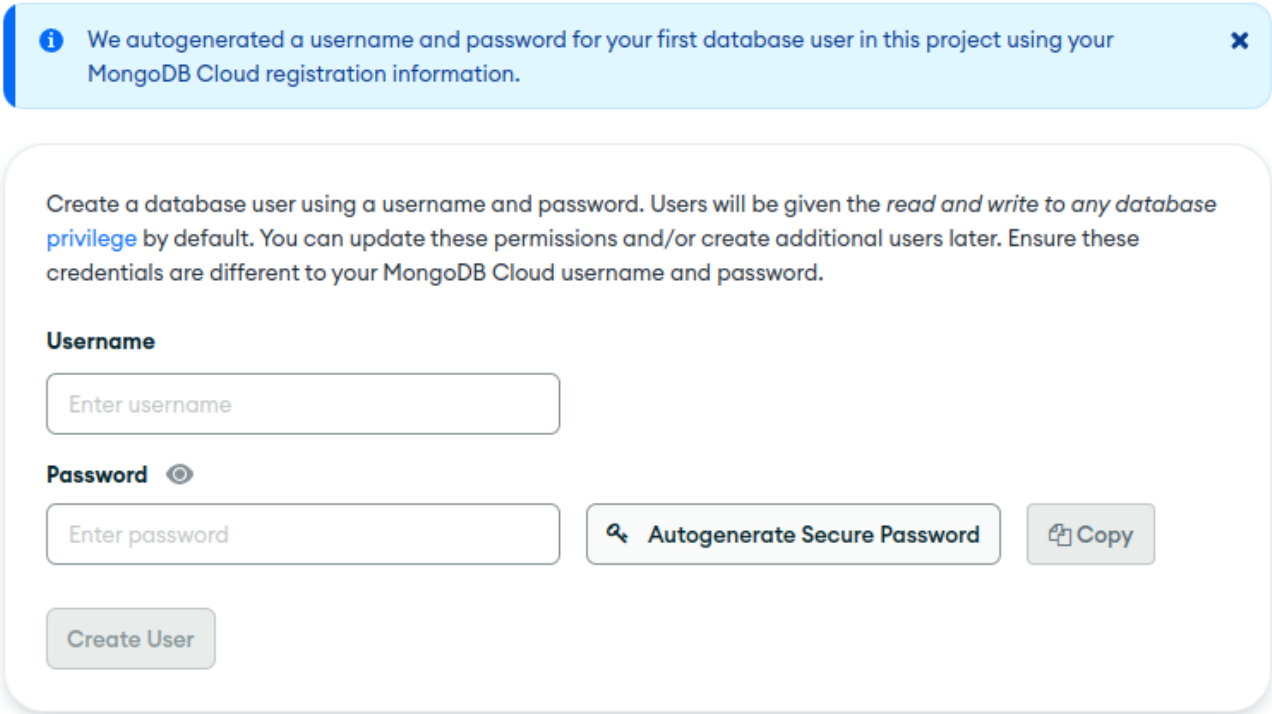
```
"license": "ISC",
"dependencies": {
  "mongodb": "^5.1.0"
}
```

Creating an Atlas Cluster and Loading Sample DB

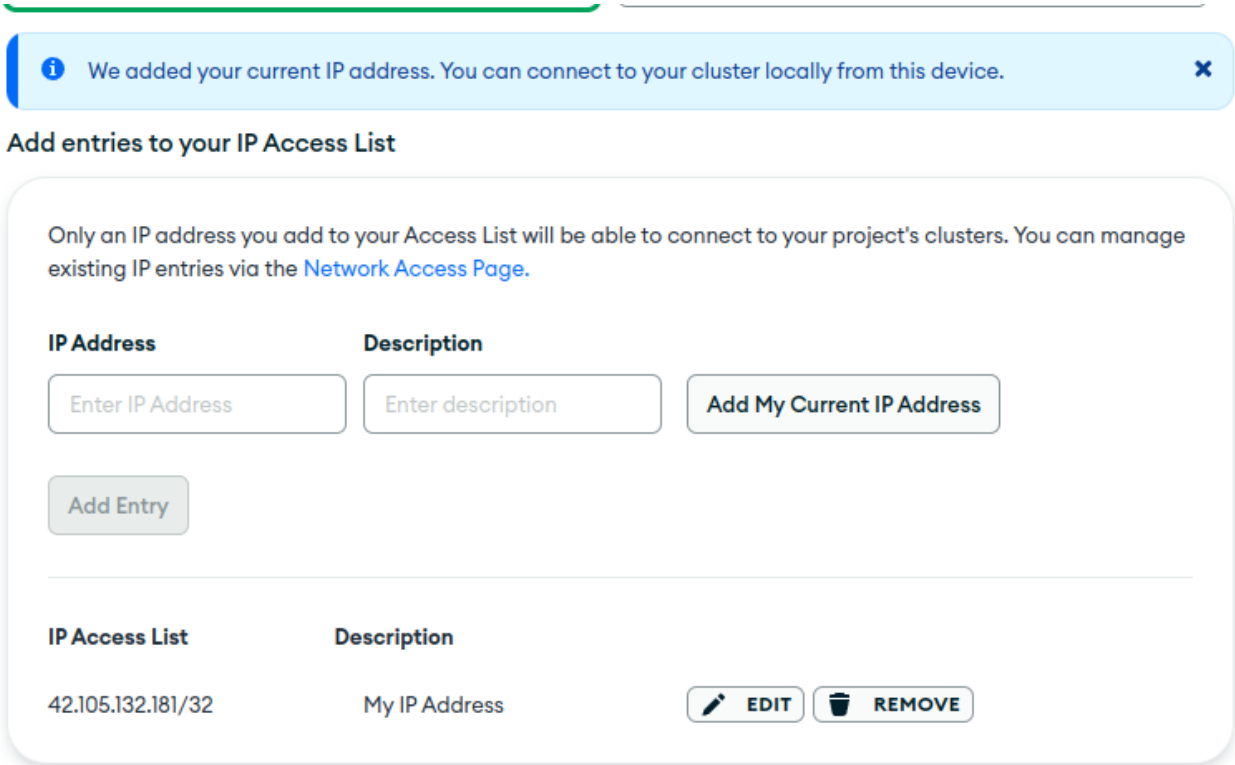
Create a new account on the website mondogb.com and then log in into it
Then hit the site cloud.mogodb.com and click on create cluster then select shared cluster and then click on the create cluster button.


Atlas
A Develops
Access Manager
Billing
All Clusters
Get Help
MERN Book App
Data Services
App Services
Charts
DEPLOYMENT
Database
Data Lake
PREVIEW
SERVICES
Triggers
Data API
Data Federation
Search
SECURITY
Database Access
Network Access
Advanced
New On Atlas
Goto
A DEVELOPS > MERN BOOK APP
Database Deployments
Find a database deployment...
+ Create
Cluster0
Connect
View Monitoring
Browse Collections
FREE
SHARED
Your cluster has been automatically paused due to prolonged inactivity.
Resume your cluster to connect to it and to gain access to your data.
Resume
VERSION
5.0.8
REGION
AWS / Mumbai (ap-south-1)
CLUSTER TIER
M0 Sandbox (General)
TYPE
Replica Set - 3 nodes
BACKUPS
Inactive
LINKED APP SERVICES
None Linked
ATLAS SEARCH
Create Index


Serverless
Dedicated
Shared
For learning and exploring MongoDB in a sandbox environment. Basic configuration controls.
No credit card required to start. Upgrade to dedicated clusters for full functionality.
Explore with sample datasets. Limit of one free cluster per project.
Cloud Provider & Region
AWS, Mumbai (ap-south-1)
aws
Google Cloud
Azure
Recommended region
Dedicated tier region
NORTH AMERICA
EUROPE
AUSTRALIA
FREE
Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.
Cancel
Create Cluster



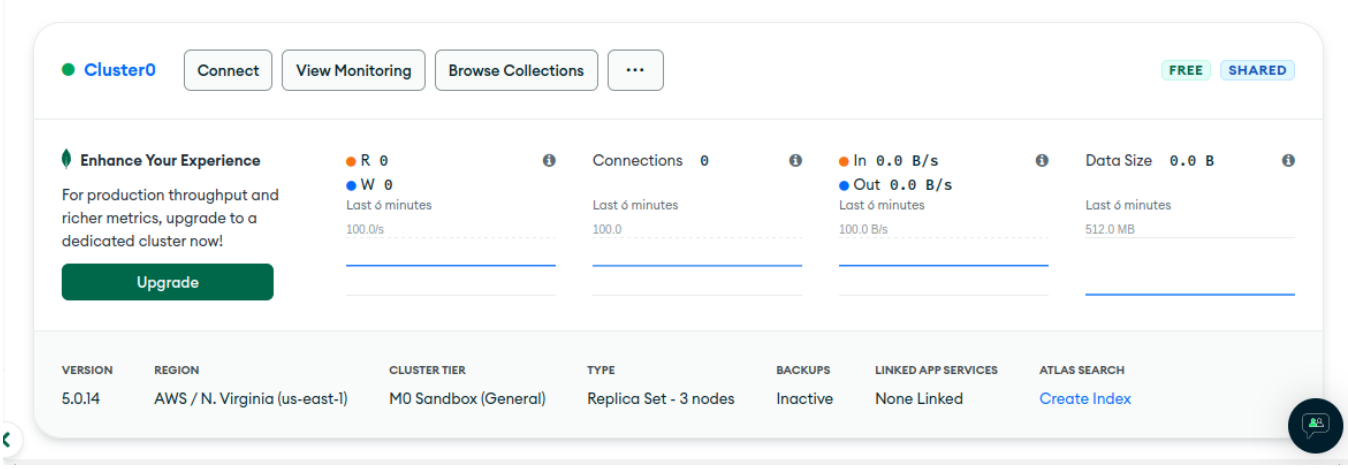
Now create a user in the cluster by providing the username and the password and clicking on Create User button, we will be using these credentials to log in to the cluster.



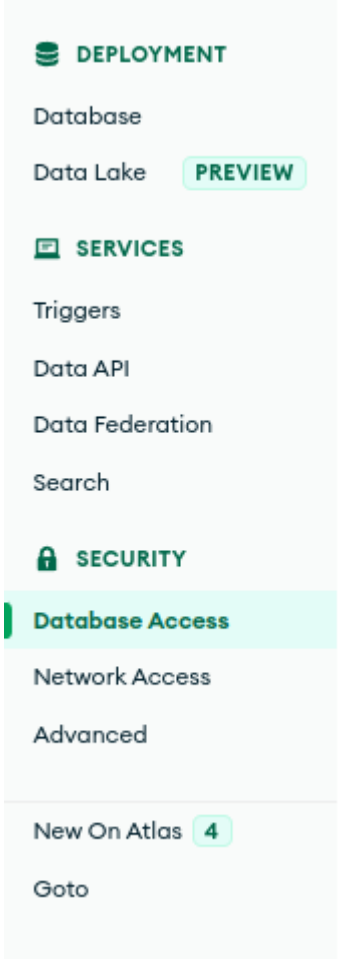
You will also have to add the IP address from which device you want to access the Database.

After this click the Finish and Close button.

After the db has been created it will look like this :

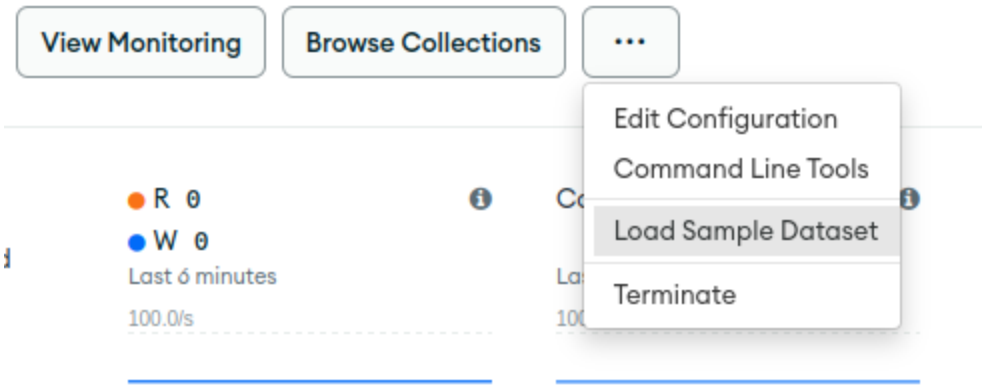


If we want to make some changes to these things we can always go to the left panel which contain the options :

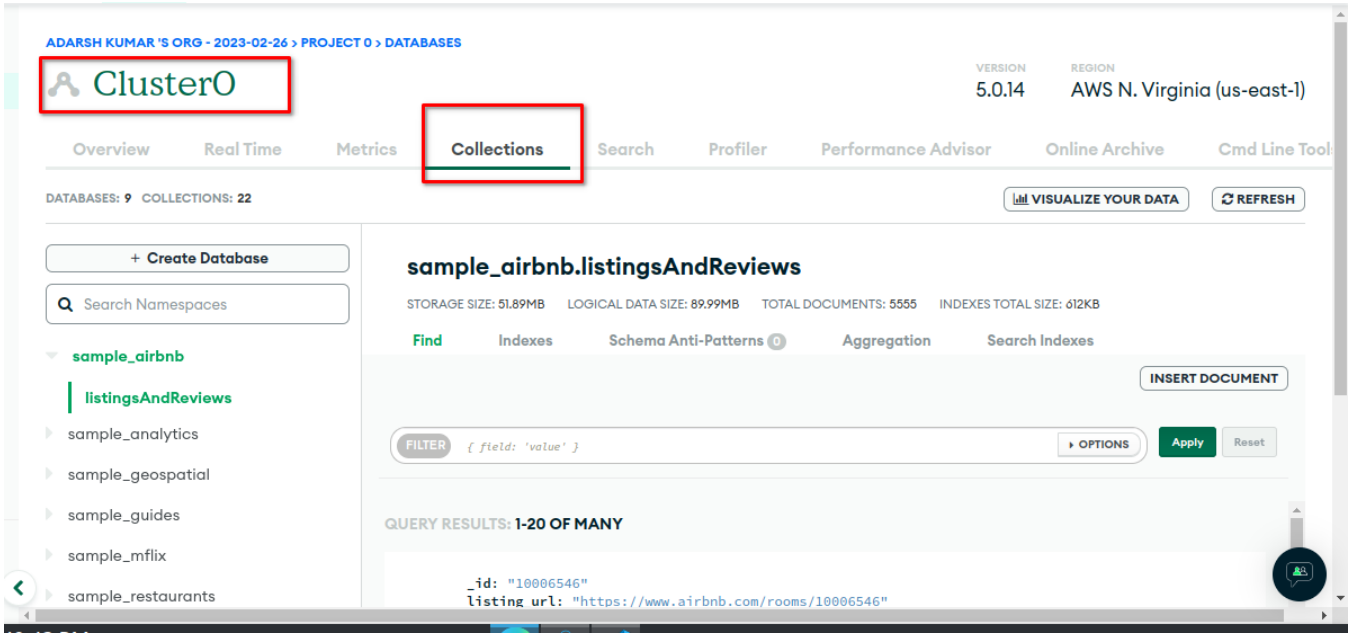


From here we can add a new user, make user admin, add ip, etc.

We will load the sample data on the cluster by :



If you open the DB and head to the Collections tab it will look like this :



Creating connection MongoDB database

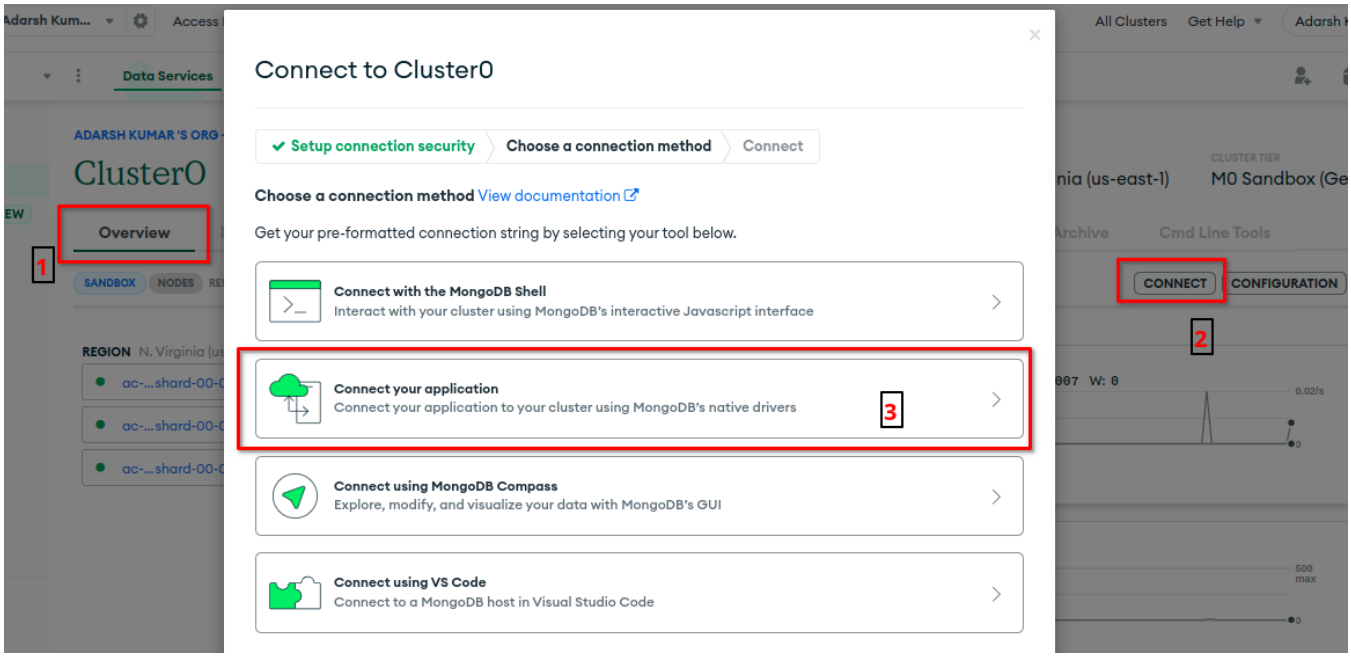
Import the MongoClient object:

After installing the MongoDB driver, you need to import the MongoClient object provided by the driver in your Node.js code. You can do this by adding the following line at the beginning of your file:
const { MongoClient } = require('mongodb');

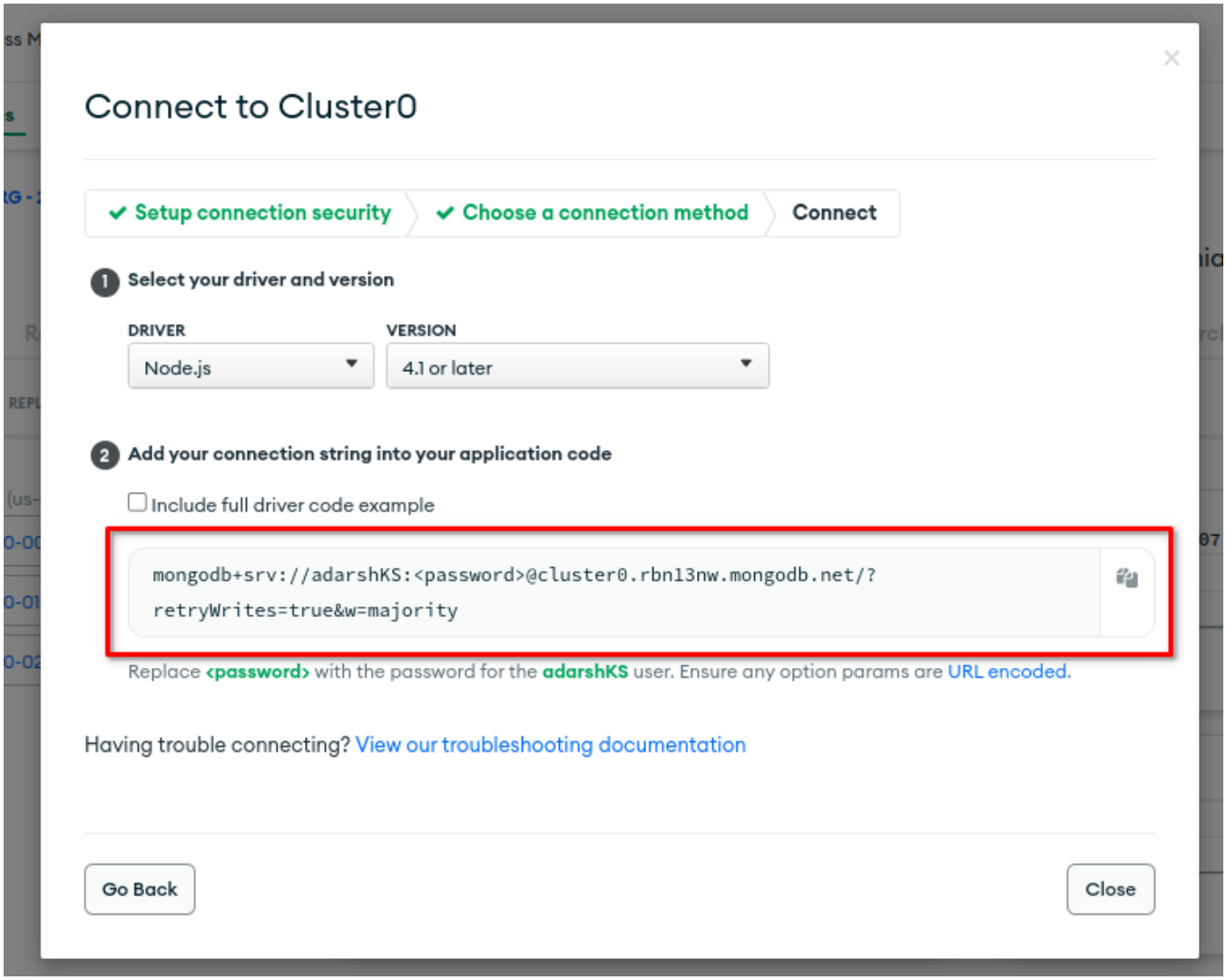
MongoDB connection URL

You can get the MongoDB connection URL by :

1. Open overview tab.
2. Click connect.
3. Click connect your application.



4. Copy the connection string from there.



We will store the uri in a string.

Here we have to replace the `<password>` with the password that we have set while creating the user
At this point the index.js looks like :

```
const { MongoClient } = require('mongodb');

const uri = "mongodb+srv://adarshKS:pODzpywpZhnv4fJd@cluster0.rbn13nw.mongodb.net/?retryWrites=true&w=majority"

```

Don't share the password with anyone.

Create a new MongoClient instance

Once you have the MongoDB connection URL, you can create a new instance of the MongoClient object by passing the connection `uri` as a parameter, like :

```
const client = new MongoClient(url, { useUnifiedTopology: true });
```

In this example, we are creating a new MongoClient object and passing the MongoDB connection URL as the first parameter. We are also passing an options object as the second parameter. The options object is used to enable the new Unified Topology engine in the MongoDB driver.

Connect to the MongoDB server

After creating a new MongoClient object, you need to call the connect() method to connect to the MongoDB server. Here's an example:

```
client.connect((err)=> {
  console.log("Connected successfully to server");
});
```

Here we are calling the `connect()` method on the client object and passing a callback function as a parameter. The callback function is called once the connection to the MongoDB server is established. If there is an error during the connection process, the callback function will receive an error object.

Note that here client refers to the current user for which the DB is being accessed.

Access the MongoDB database

Once you have established a connection to the MongoDB server, you can access the database using the client object like :

```
const db = client.db('sample_airbnb');
```

Here we are calling the db() method on the client object and passing the name of the database that we want to access. This will return a new Db object that we can use to perform various database operations.

Here we will be printing the names of all the databases available in the cluster so we will be using the chain of functions like :

```
const dbList = client.db().admin().listDatabases()
```

Close the MongoDB connection

After you have finished performing database operations, you should close the connection to the MongoDB server like :

```
client.close();
```

Here we are calling the `close()` method on the client object to close the connection to the MongoDB server.

Finally, we can do some exception handling by using try-catch-finally block, and make the code modular by placing them in different functions like :

```
const { MongoClient } = require('mongodb');

const uri = "mongodb+srv://adarshKS:pODzpywpZhnv4fJd@cluster0.rbn13nw.mongodb.net/?retryWrites=true&w=majority"
const client = new MongoClient(uri, { useUnifiedTopology: true });

1 reference
async function listDatabases(client) {
    dbList = await client.db().admin().listDatabases();
    console.log("avaibale db are : ");
    1 reference
    dbList.databases.forEach(item => {
        console.log(' - ', item.name);
    });
}
1 reference
async function connect_to_Mongo() {
    try {
        await client.connect();

        await listDatabases(client);
    } catch (e) {
        console.log(`error: ${e}`);
    }
    finally {
        await client.close();
    }
}

connect_to_Mongo();
```

Here the function `connect_to_Mongo()` will be used to make a connection to the mongodb cluster. Since all of these are asynchronous functions, we are using await to wait until we get some response. On running the indes.js file we get the following in the terminal.

```
>> node index.js
avaibale db are :
 -  sample_airbnb
 -  sample_analytics
 -  sample_geospatial
 -  sample_guides
 -  sample_mflix
 -  sample_restaurants
 -  sample_supplies
 -  sample_training
 -  sample_weatherdata
 -  admin
 -  local
```

We can also catch the error from the `connect_to_Mongo()` function by using the catch function and calling the `console.error()` function like :

```
connect_to_Mongo().catch(console.error());
```