Adding a Database

In this lab, we're going to look at databases and logins. You'll be setting up a database server on your Linux server or you can try setting it up on Windows.

At the end of this lab you should have a working local web system connecting to a database that could run on both your Windows and Linux machines, both using the same code base (on GitHub). You should also be able to serve database and web server on different computers.

The other key outcome from this lab is that you *should* be using environment variables.

Before you begin….

GitHub

You will want a GitHub repository for this lab. You can use the one you used in the previous labs.

In either case, you should start with a server .js file that used express and package.json file that installs the node_modules folder. You do not need to include node_modules in your GitHub.

Databases

You have the option to set up and use 2 databases for this lab:

- One on the virtual Linux box
- One on the local Windows machine (assuming software is available)

For your web pages, you need two databases:

- A "dev" database for development
- A "live" database (which would be the "real" database in a real application, one you probably don't want to mess up)

In the first instance, both of these databases will be mongo databases, and your website code will need to know **a url** to access them which you will set with an environment variable.

At the end of this lab, there are also some instructions for configuring a cloud-based MongoDB that lives in MongoAtlas. Unfortunately, your websites (running inside RGU's firewall) will not be able to access MongoAtlas (running outside the firewall) because it does not use http or https as the access protocol/ports. But the option is there because to use MongoAtlas, you need to set up the same security as you do running a database locally. And to show that you could switch from a local "dev" database to a "live" database simply by changing one (environment) variable.

Mongo is good for a first database because it's a non-relational (NoSQL) database, so you don't need to define any kind of schema before you use it (but we will use mongoose and define a schema anyway).

Installing the database tools (on Ubuntu)

If you are using the RGU Windows computer, then mongod and mongosh should already be installed. Check this using PowerShell and type **mongod -v** and **mongosh -v**.

To run a database on your Linux box, you are first going to have to install the MongoCommunity tools. We're going to use two command line tools: **mongod** and **mongosh**.
mongod – mongo **d**atabase (runs a database)
mongos – mongo sharded database (not expressly covered here, but runs a sharded database)
mongosh – mongo **shell** (accesses a running database and lets you edit it)

Using the following links, you should be able to set up a locally hosted Mongo database on your linux box. The mongod server and installation instructions are available from here https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/#std-label-install-mdb-community-ubuntu

This should install **mongod** and **mongos**. Check that they're installed with **which**
Note that this tutorial gives details of running mongod using a daemon. This is a good idea, particularly if you are running on a computer you are accessing with only one terminal (e.g. an AWS instance accessed by ssh), but in the first instance, we'll run mongod without a daemon (because it lets you see all the output).

You also need mongosh. You can install mongosh from:
https://www.mongodb.com/try/download/shell
(if you download the .deb option, it can be installed with **sudo dpkg -i <downloaded_file.deb>** putting in your own filename)

Check they've installed using **which** and their help/version options.

Your local database

We're going to set up a local database that will run locally on your "local" Linux (or Windows) machine. It's "local" because you'll also be running the web server on that machine. We do this using Mongo Community tools (that you've just installed). In the first instance, this is going to serve as your development server (i.e. the one you can mess up).
There's a good guide to the commandline tools here:
https://www.helenjoscott.com/2022/01/29/mongod-mongo-mongosh-mongos-what-now/.

There are two tools:
- mongod (for setting up databases)
- mongosh (for looking at them).

In addition, there is MongoCompass if you want a GUI for looking at the databases.

You can check your environment for the mongo daemon by typing:
```
mongod --help
```

Now create a folder (directory) for your database to live in (**mkdir** from the commandline) and initialise your database using:

```
mongod --dbpath <path-to-your-database-folder>
```

Alternatively, go to the directory where you want your database to live (using **cd**) and just type **mongod --dbpath .** which will set up the database in the current folder (which is what the single dot means). Note that this command will occupy that terminal (or PowerShell), so you'll want to start a new terminal (or a new tab – there's a + button to click) and leave that one running. Later instructions involve running mongod within its own daemon – that will give you your command prompt back.

Make sure you put the right path to your database folder. If you look, you'll see it fill up with stuff. All these weird files can be interpreted by mongosh or MongoCompass. Now you have a database server up and running. By default, the database will be running on **127.0.0.1:27017** (or localhost:27017) and you can check this either in your browser (which will tell you something is being served but you're connecting to it oddly), or by using the mongo shell (mongosh) or by attaching MongoCompass.
The mongo shell comes from https://www.mongodb.com/docs/mongodb-shell/install/ to install it, but you can also use MongoCompass to check on your database.
Type:

```
mongosh --host 127.0.0.1:27017
```

into the new terminal to access the mongo shell for your database.
Ctrl+C (or type **exit**) for when you need to get out of the mongo shell, but first from within the mongo shell, you can type:

```
show dbs
```

to show the databases that live in your folder. Right now, there are only default databases (admin and config and local), so you won't see much. We'll come back to the mongo shell later, though. If you want to take a look through MongoCompass, you are welcome to view your mostly empty database.

If you want to stop your local database (mongod), ctrl+c will kill it in the command line where it is running (unless you've started it up with an **&** at the end, in which case you will have to locate the relevant process with **ps** and kill it with **kill**). Remember that **you will need your database running to access it**.

Back to the web server – a "nicer" front end and an API call

For an example on using the database, you can look at "index.js" from Lab 1&2 (our simple Node server that already has some basic routing in it). Remember, to run it, you need to type **node index.js** at the command line in the same directory as index.js (or from the directory that you usually call it from. You can then type various urls in the browser address bar and see various text messages in return (depending on what each function does).

We are going to add some functionality to the front end so that we can store a "quote" for a web service. We'll give the quote a name, and take some of the input parameters to store the quote in the database.

You should be able to modify the html code to add a text input for "quoteName" and a button to call a backend API function to store the quote. The function from your button should take some details from the page and store them (exactly which details it takes is up to you). You will need to name your API call and add it to the back end (index.js), but you can mostly copy your previous API function for this. In the first instance, simply take the input from the front end, read it, log it to the console and return the quote name as a response. This is just a temporary thing, we'll add them to the database next. Make sure your basic API call works (i.e. your served front end can push the button, you see some console output on your server side and a response on your front end). If you are stuck, look at the possible solution and lift the UI code from there, or nicely ask an LLM to generate the code.

Getting to the database

We're going to edit code in this section. If you're staring blankly at your Linux box wondering how to open or edit the code on there, there are a number of options:

- Windows -> Git -> Linux (i.e. only edit on your windows machine, push to GitHub and then pull the code down to Linux, never edit on the Linux machine) This ensures your code is backed up at all times, but it gets old fast when you are debugging every little thing via GitHub.
- Install **gedit** or **nano** on your Linux machine and learn how to use them (nano should already be installed). They're just text editors. nano works via a terminal and is a lot nicer to use than vim which comes installed by default on Linux.
- Choose your own IDE to install on your Linux machine.

First, we're going to access the local database in our dev environment. In your backend server file (index.js?), set up the MongoClient according to the instructions given at https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/connect/#std-label-connect-atlas-node-driver (or as described below).

At the top of the file (with the other const declarations) require the node package like this:

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

Close by, set up a uri for your database like this:

```
const uri = "mongodb://127.0.0.1:27017";
```

This is the uri for the local development server. ***If*** you were using a MongoAtlas server, you'd use the uri you obtained from the MongoAtlas Wizard (which begins **mongodb+srv**), as described below. But you won't be able to access the MongoAtlas server from within RGU's network because of the firewall. If you were using a database on a different computer, you'd have a different uri. And if you had some security settings on your firewall, you'd have a different uri. So, take note how this uri is constructed with the callback IP address and port number. You might be able to use localhost instead of 127.0.0.1.

The code that you need to connect to the database is:

```
  // Database stuff
  // Create a new MongoClient
  const client = new MongoClient(uri);
  async function run() {
  try {
    // Connect the client to the server (optional starting in v4.7)
    await client.connect();
    // Establish and verify connection
    await client.db("admin").command({ ping: 1 });
    console.log("Connected successfully to server");
    console.log('Start the database stuff');
    //Write databse Insert/Update/Query code here..
        console.log('End the database stuff');

  } finally {
    // Ensures that the client will close when you finish/error
    await client.close();
  }
  }
  run().catch(console.dir);
```

You *can* put that in your server if you want (in a suitable api function that you can call from the front end). It checks that a connection can be made to the database defined by the uri and that the “admin” database (which is present by default) can be pinged. Adding the above code to the API call that does the database stuff will check that you can access your database.

However, the code that you *actually* need to connect and insert one item is:

```
  // Database stuff
  // Create a new MongoClient
  const client = new MongoClient(uri);
  async function run() {
  try {
    //Write databse Insert/Update/Query code here..
    var dbo = client.db("mydb");
    var myobj = { quoteName: n, salary: s, days: d }; //******CHECK!!!****
    await dbo.collection("quotes").insertOne(myobj, function(err, res) {
      if (err) {
        console.log(err);
        throw err;
      }
      console.log("1 quote inserted");
    });
    console.log('End the database stuff');

  } finally {
    // Ensures that the client will close when you finish/error
    await client.close();
  }
  }
  run().catch(console.dir);
```

This code assumes that you have set your api call up with variables n, s and d (representing quote name, salary and number of days, respectively) taken from the API query parameters. Your code may be different but remember to extract your parameters from the query. In my code, I am storing the quote name, the salary and the number of days in a database called **mydb** and in a collection called **quotes**. I log any successes to the server console.

You can try to run this with **node server.js**. Look at the server console to make sure it connected to the mongo database correctly. If there are errors, read and correct them.
It might not work!
You might need to use **npm install mongodb** to locally install mongodb for node (and add it to your package.json file with the --save option).
You might need to start your local database server (possibly in another terminal/PowerShell window).
You might need to upgrade node.js so it's running the most recent version (see last week's lab regarding this – look for "curl" in the lab sheet).
You might not have any security on your database, in which case you will have no username/password combination in the mongodb url.

You should now have some Node/Express server code that takes user input via a web page and writes it to a database. Do test it locally to make sure it works. You can do this by entering the values in the basic html page, or you might formulate a curl command to make a suitable http request to your server, or you can use Postman if you prefer that to curl. You can use the Mongo Shell to view entries in your local database. Mongo shell opens with:

```
mongosh --host 127.0.0.1:27017
```

and you can look up some mongo shell commands like the ones below to see what is in your database (remember to replace anything in <angled brackets> with your own information).

```
show dbs
use <mydb>
show collections
db.<myCollection>.find()
```

With my code, I find a database called **mydb** and in it is a collection called **quotes**.
Once your server code is writing to your database, examine your database with the mongo shell (or with MongoCompass) and see if you can see your database additions.

If you scroll down this lab to the section on "Database Security: Notes on mongosh and mongod", you will see that there may be occasions when you wish to have a password form part of the URI for your database.

Basics Completed

You've now managed to set up, configure and add a **locally hosted** MongoDB to your Linux-running server. You can save user input to a local database.

Database Security: Notes on mongosh and mongod

With a very simple mongo server startup (**mongod --dbpath .**), the following warnings are generated within mongo shell:

```
2023-02-06T17:34:37.572+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2023-02-06T17:34:37.572+00:00: This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable this warning
   2023-02-06T17:34:37.572+00:00: Soft rlimits for open file descriptors too low
```

There are a number of things we need to do to make the database both secure and accessible. These also correspond to things that need to be done in MongoAtlas, however there, they are done using the web interface or MongoAtlas dashboard. The things that need sorted out are:

- Access control (user accounts)
- IP restrictions (in the first instance, allowing all IP addresses with --bind_ip_all is a suitable first step but not before you add user accounts.

Right now, the database is running on localhost only, so it is actually completely secure. Which is good because if it was open to all IP addresses, it would be completely open (with no authentication necessary).

Access control in the database

Details about creating a user in the database via mongosh are given here:
https://www.mongodb.com/docs/manual/tutorial/configure-scram-client-authentication/
for creating the super user. And here:
https://www.mongodb.com/docs/manual/tutorial/create-users/
for creating "normal" database users. Your web app is a "normal database user".
You can do all this via mongo shell (or mongosh).
First, you should create a superuser for all your databases (in **mongosh**):

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: passwordPrompt(), // or cleartext password
    roles: [
      { role: "userAdminAnyDatabase", db: "admin" },
      { role: "readWriteAnyDatabase", db: "admin" }
    ]
  }
)
```

You will be prompted to set the password for your superuser (because you used the passwordPrompt() function. Shut down your database, and restart it using:

```
mongod --auth --port 27017 --dbpath .
```

This assumes that you are starting the database in the folder where you want the database to live.
Next, you need to create some users (in mongosh).

```
use mydb
db.createUser(
  {
    user: "testUser",
    pwd:  passwordPrompt(),   // or cleartext password
    roles: [ { role: "readWrite", db: "mydb" } ]
  }
)
```

Again, you will be prompted to set a password at the prompt. The above code uses a database called mydb and it creates a user called test who has read/write access to the database mydb. In my case, I gave the user the password password. It's not exactly secure.

If you try to access your database from your server now, you will need some more information in your uri. https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/connect/#std-label-node-connect-to-mongodb This link shows you how to construct the uri for MongoAtlas, but for a server started by mongod community server, the uri would be:

```
const uri = "mongodb://test:password@127.0.0.1:27017/mydb";
```

Where the database is called *mydb*, the user is called *test* and the password is *password.* Our database is not a sharded database (like the ones used on MongoAtlas), so no need for +srv in the uri. Our user only has access to one specific database so we need to access that one specifically (hence the uri containing "mydb").

Environment Variables

You may have used these before when you set the port number in a server. This time, you will be using them to set the database uri. Using Environment Variables will help to keep your private things (like database passwords) out of GitHub.

Within your server code (index.js), you use environment variables like this:

```
const DBURI = process.env.DBURI;
```

or

```
const PORT = process.env.PORT || 8000;
```

You then simply use your const value in place of any magic numbers or magic strings. This is good coding practice.
In some PaaS environments (e.g. Heroku), an environment variable (PORT) sets some things automatically. For this to work, you'll need to set the equivalent environment variable in your local environment, too. One way you can do this is via the command line. For example, if you have env.process.DBURI referenced in index.js (or whatever your node file is called), then you could type **DBURI="<my_db_uri>" node server.js** at the command line to set the environment variable on the fly, or you might store that in a script. On a Linux machine, environment variables can be set up on your user profile in .bash_rc. Alternatively, they can be set up inside shell scripts for running the code. Or, see the docs for PowerShell: https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_environment_variables?view=powershell-7.2 but remember that the environment variable may last only as long as your PowerShell session. If you close a terminal, any variables set within that terminal disappear (except if they're written in .bashrc which is ran every time a new bash terminal is initialised. You might want to look at creating a script file for setting up a virtual environment with virtual environment variables on the fly.

Once you have your server running with your database, you can check to see if it works on a local browser, but you can also check to see that it works with a remote browser (i.e. work out the URL for your virtual Linux computer and try accessing it with another computer attached to RGU's network. It should be possible for you to load your website on your phone if it is connected to eduroam.

Lastly, you should check all your server code into GitHub.

The Scripting you should learn

Git clone (or pull) your (up-to-date) repository from github
Install the appropriate node modules (**npm install**).
Start up the database (look at how the daemon works once you have everything up and running)
Firewall holes (from last lab)
Start the server up (again, look for nodemon or some sort of daemon to run your node)
And if you really want to be complete – environment variables, installing up-to-date applications.

What do you mean, look at last lab?

Yeah, putting it here to allow you to be a virtuous programmer and enhance your lazy skills.
If you have trouble with **npm install mongodb**, then you might need to upgrade your node version (this was in the previous lab, but repeated here). By default, Ubuntu installs a very old version. The official way to update is to do this (secondary reference from here https://askubuntu.com/questions/426750/how-can-i-update-my-nodejs-to-the-latest-version dayuoli's answer):
curl -fsSL https://deb.nodesource.com/setup_current.x | sudo -E bash –
(optionally restart terminal here)
sudo apt-get install -y nodejs
That retrieves a script to update the ubuntu repositories to point to the current node repositories (instead of the repositories that contain node version 12.22.9). It then installs nodejs from these repositories (the -y flag selects yes for any options given).

What about Windows?

If you can find versions of mongod and mongosh to run on Windows (they're installed on RGU computers), then the instructions for setting up a database are the same as for Linux (but you'd run them through PowerShell instead of a bash terminal). Alternatively, see if you can find a way to run a database from MongoCompass.
You can then try any combination of:

- Run server *and* database on Windows
- Run server *and* database on Linux
- Run server on Linux and database on Windows
- Run server on Windows and database on Linux

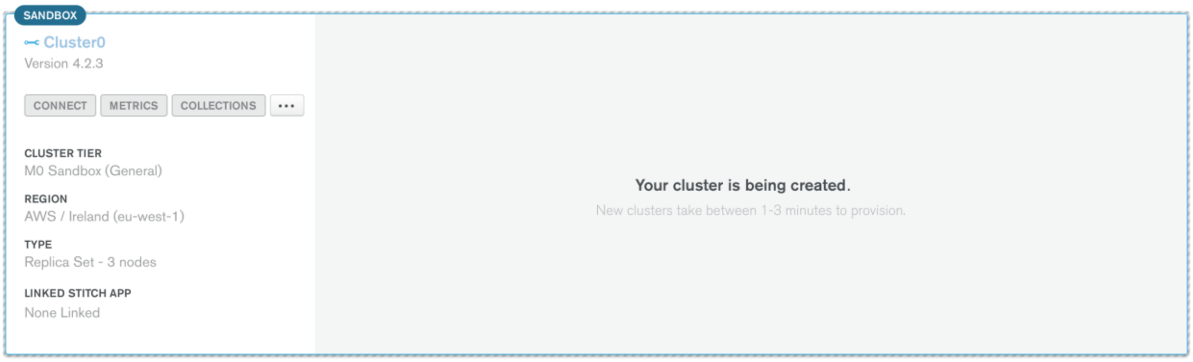
The only thing you should change in these is the database uri. Everything else should work, but debugging any issues will give you some valuable experience. Remember to poke a hole in ubuntu's firewall to let the database port 27017 out.

Supplemental MongoDB in MongoAtlas

There is a free online MongoDB available at https://www.mongodb.com/

Only continue if you are outside of RGU's firewall!!!!!!!!!!!!

Go ahead and sign up and log in using the web interface. It won't need a credit card for the free tier stuff. Create a "cluster". A cluster is just Mongo's way of saying a place to store databases, but with some n-tiered architecture built in (e.g. backups, or sharded access to speed up database reads/writes). You can create a sandbox cluster, select AWS (because it has free tier options) and select an appropriate location. If in doubt, select the free options.



CLUSTER TIER
M0 Sandbox (General)

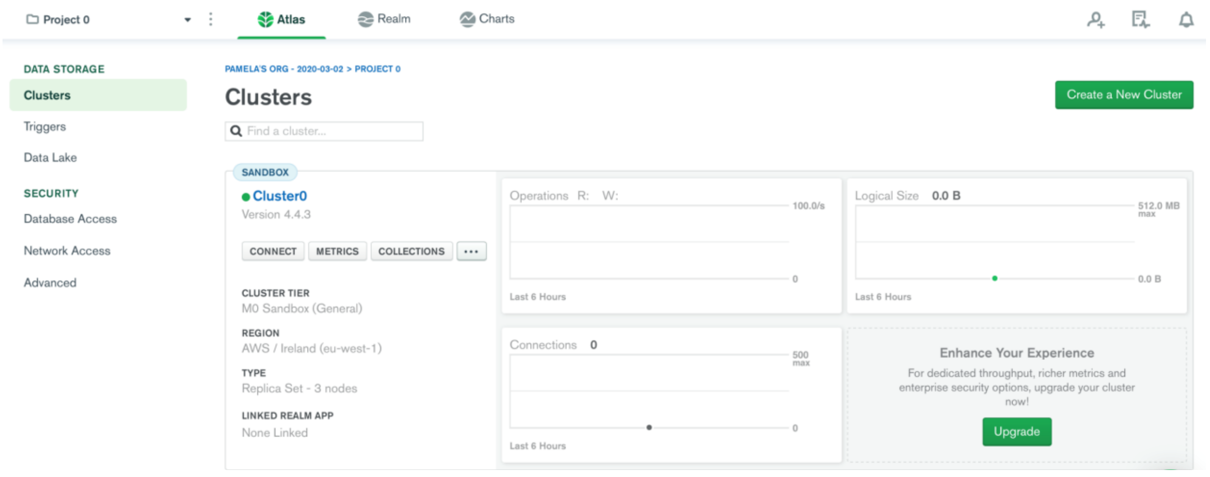
REGION
AWS / Ireland (eu-west-1)

TYPE
Replica Set - 3 nodes

LINKED STITCH APP
None Linked

Your cluster is being created.
New clusters take between 1-3 minutes to provision.

It'll take a few minutes to create your cluster but once it's done, you'll see something like this:



CLUSTER TIER
M0 Sandbox (General)

REGION
AWS / Ireland (eu-west-1)

TYPE
Replica Set - 3 nodes

LINKED REALM APP
None Linked

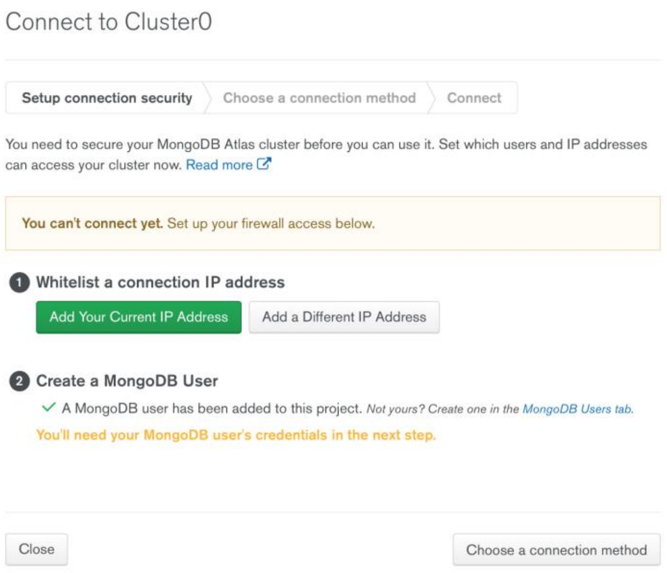
Enhance Your Experience
For dedicated throughput, richer metrics and enterprise security options, upgrade your cluster now!

Now, you'll need to add some user access to it. Go in to "Database Access" and add a user (with a password). Remember to give them read and write access. Select a secure password but pay attention to the "special characters" notes – you'll have to type the password into a URL and you'll have to HTML encode special characters (your browser may well do this for you).

It's worth noting that your user is added to the "Project" not the database, so if you delete your database and start again, any users you create here will still exist.

You should keep these user access details secure (i.e. NOT in your public GitHub). But remember that **you** can change them at any time.

Now you have to set up your connection to your cluster in Network Access:


Connect to Cluster0

Setup connection security | Choose a connection method | Connect

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

You can't connect yet. Set up your firewall access below.

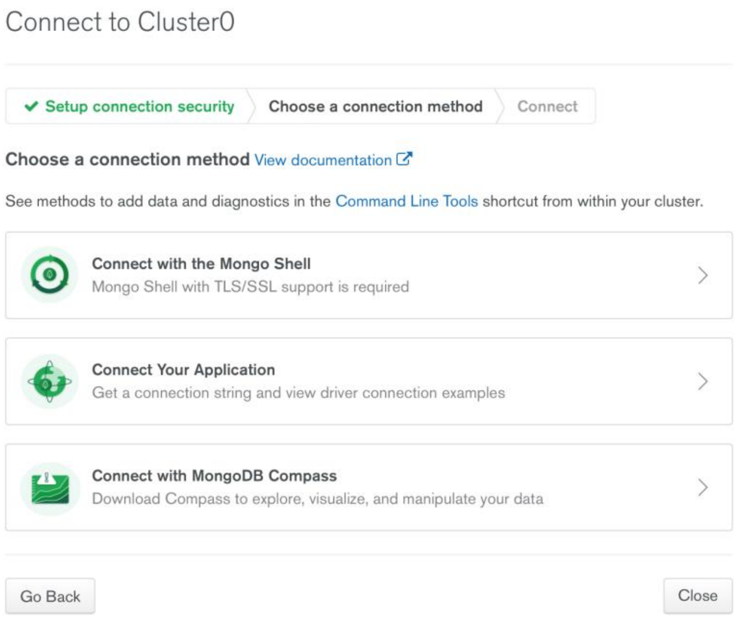
1 Whitelist a connection IP address

2 Create a MongoDB User

✓ A MongoDB user has been added to this project. *Not yours? Create one in the MongoDB Users tab.*

You'll need your MongoDB user's credentials in the next step.

You already have a user, so you can whitelist the current IP. If you have any trouble at all, simply whitelist all IPs. It's not as secure, but if you do not have a static IP address and cannot figure out your range of IP addresses on your cloud server, it's your best bet.


Connect to Cluster0

✓ Setup connection security | Choose a connection method | Connect

Choose a connection method View documentation

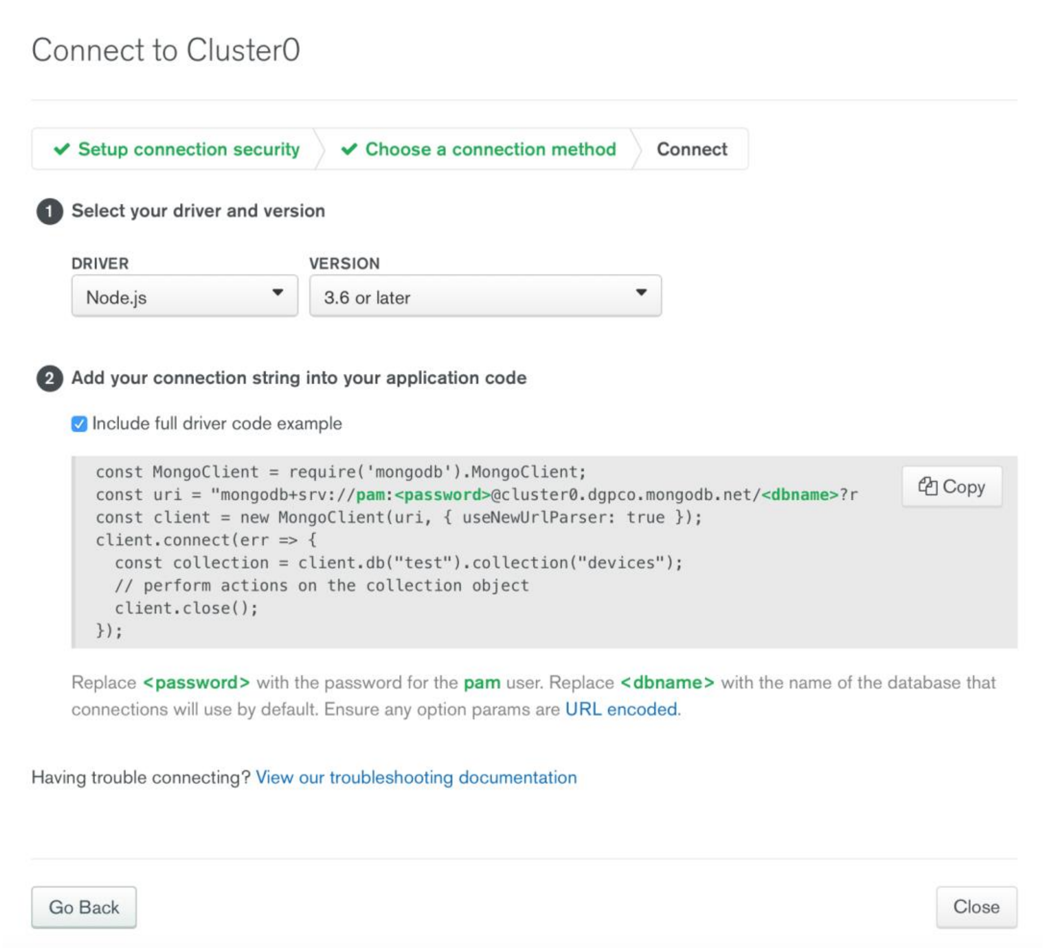
See methods to add data and diagnostics in the Command Line Tools shortcut from within your cluster.

Connect with the Mongo Shell
Mongo Shell with TLS/SSL support is required

Connect Your Application
Get a connection string and view driver connection examples

Connect with MongoDB Compass
Download Compass to explore, visualize, and manipulate your data

Select "Connect Your Application". If you have a cloud instance of your server, you can retrieve the correct url for the database to use in a Node.js server.



Connect to Cluster0

✓ Setup connection security ✓ Choose a connection method Connect

1 Select your driver and version

DRIVER: Node.js

VERSION: 3.6 or later

2 Add your connection string into your application code

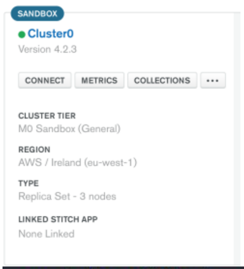
☑ Include full driver code example

```
const MongoClient = require('mongodb').MongoClient;
const uri = "mongodb+srv://pam:<password>@cluster0.dgpco.mongodb.net/<dbname>?r
const client = new MongoClient(uri, { useNewUrlParser: true });
client.connect(err => {
  const collection = client.db("test").collection("devices");
  // perform actions on the collection object
  client.close();
});
```

Replace **<password>** with the password for the **pam** user. Replace **<dbname>** with the name of the database that connections will use by default. Ensure any option params are URL encoded.

Having trouble connecting? View our troubleshooting documentation

You can copy the full driver example, but don't worry if you forget. You can call up this wizard again by going to your cluster and hitting the "Connect" button. We will use this code later in the lab.

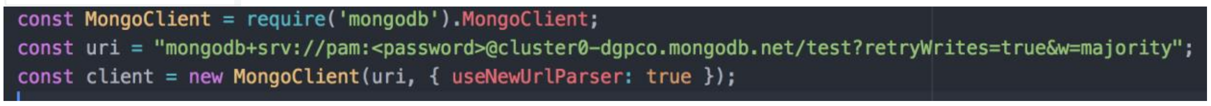


CLUSTER TIER: M0 Sandbox (General)

REGION: AWS / Ireland (eu-west-1)

TYPE: Replica Set - 3 nodes

LINKED STITCH APP: None Linked



```
const MongoClient = require('mongodb').MongoClient;
const uri = "mongodb+srv://pam:<password>@cluster0-dgpco.mongodb.net/test?retryWrites=true&w=majority";
const client = new MongoClient(uri, { useNewUrlParser: true });
```

For now, look at the code from the full driver example (extract above). You'll see that it comprises of some constants followed by some client code. We're going to set up the client code slightly different from that code, but you need to understand the uri constant. Note that your uri has two <optional> parts that you need to fill in: your password AND your database name. You can change your user name, too. In the picture above, I've called my database "test" and I haven't filled in my password. When you use the mongo atlas uri, you will need a name for your database and you need to know your password for the user you created.