Tutorial: MongoDB Atlas Setup

MongoDB Atlas is a *database-as-service* application that allows us to set up and use a basic database on remote cloud servers. It simplifies the process of managing a database server by letting someone else manage it for us! They charge a fee for bigger databases with more traffic, but only the free starter plan will be necessary for this course.

Step 1: Sign Up

Navigate to https://www.mongodb.com/products/platform/atlas-database
and click the **Get Started** button. Sign up for an account. (Note that your private information may be stored on servers outside the country and therefore subject to their privacy laws and accessible by their authorities. Feel free to use a pseudonym if you wish, or talk to me for alternatives.)

You'll be asked to verify your email address. Go ahead and do that.

You may be asked to answer a few questions about how you intend to use the service. I don't think the answers are particularly important, so just use your best judgement.

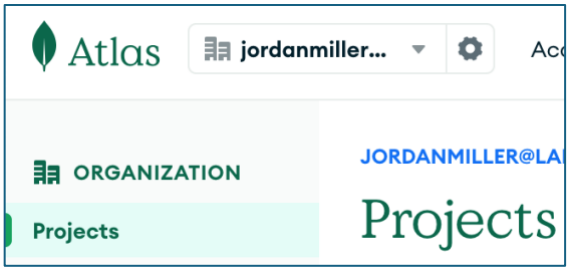
You may also be asked to choose a pricing tier. Make sure to choose the **Free** tier!

As this point, you may automatically directed to create a **Project.** If so, go ahead and give your project a sensible name (maybe "Langara") and keep the default settings for everything else. If you're not directed to create a project, skip to **Step 2** in this tutorial.

You may be automatically directed to create a **cluster.** If so, skip to **Step 3** in this tutorial.

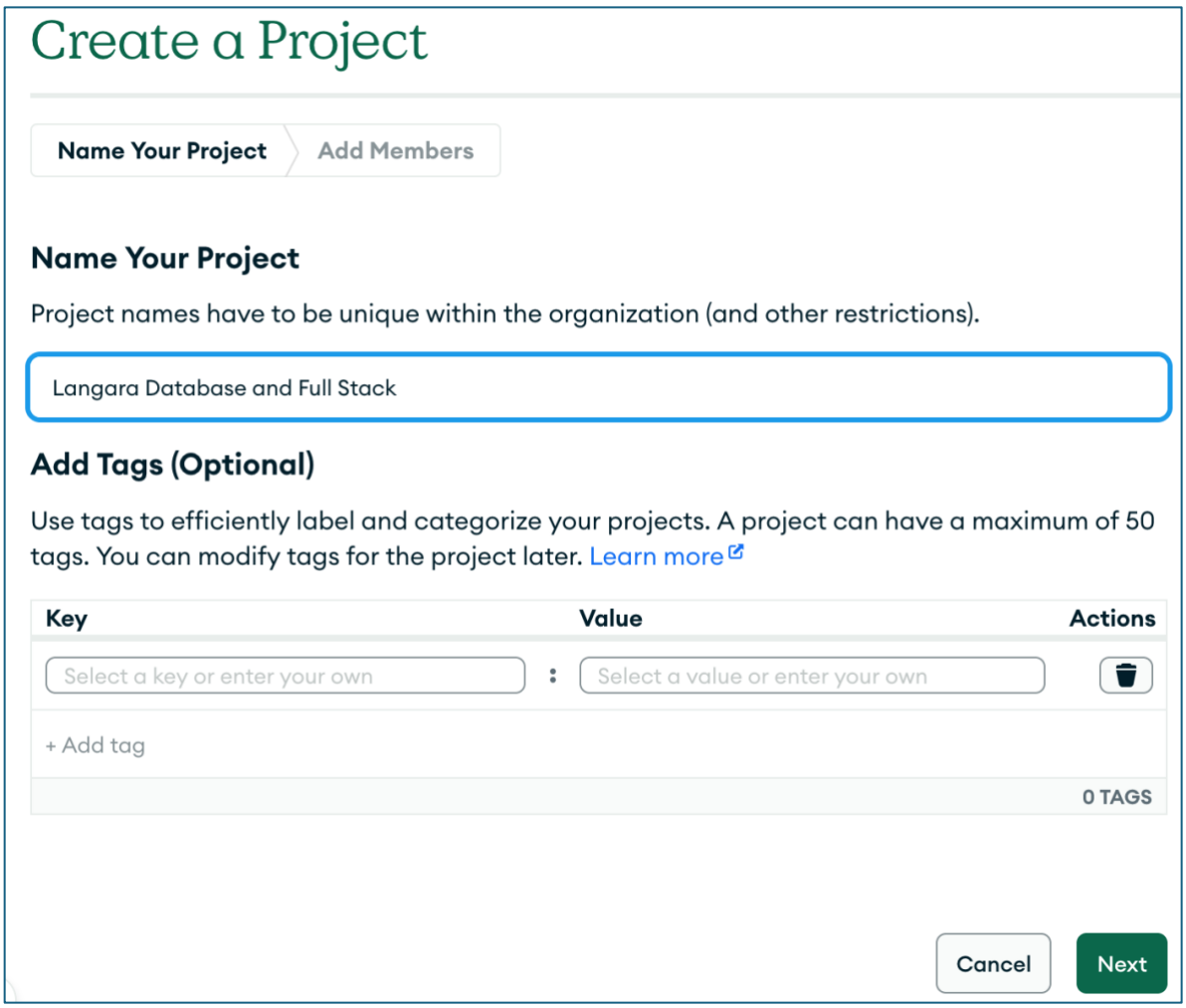
Step 2: Create a Project

In the dashboard, navigate to **Projects** in the menu bar on the left:



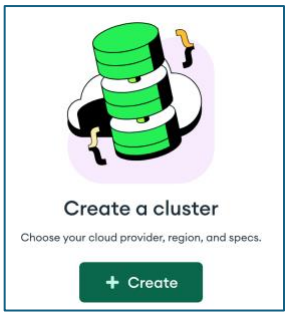
Then click the **New Project** button on the right.

Give your project a sensible name (all your databases for this class will be stored here) and click **Next:**



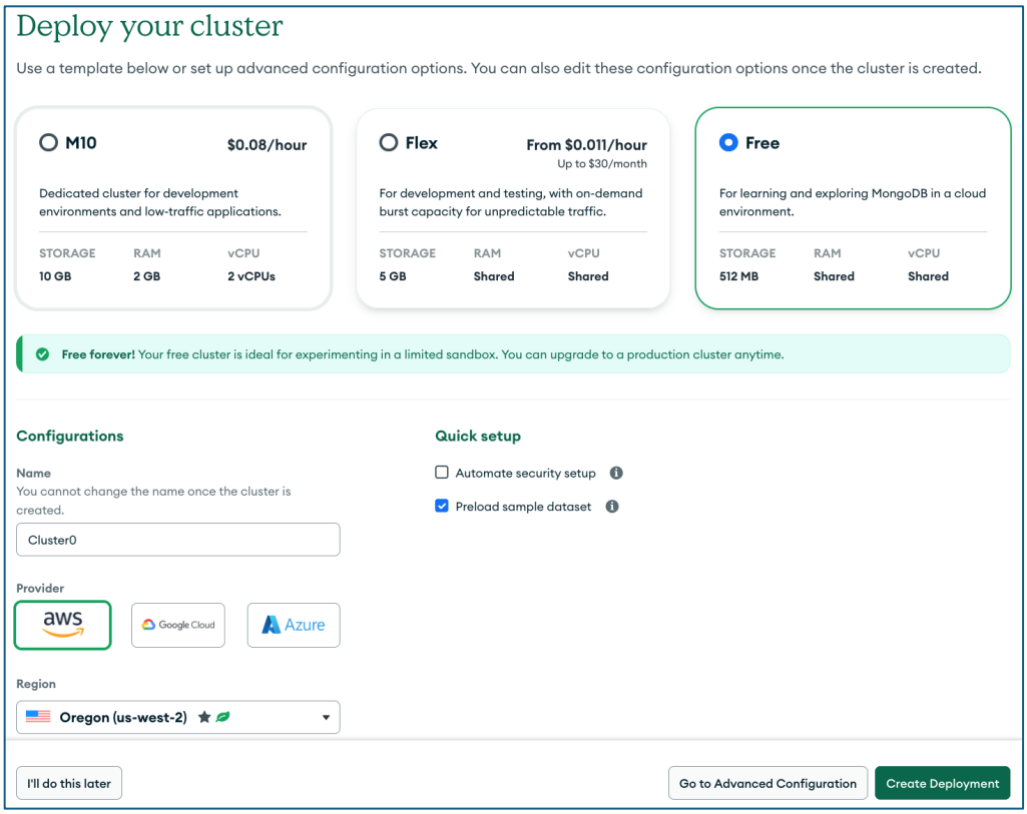
Then click **Create Project.**

You'll be taken to a page where you're prompted to create a **cluster**. Go ahead and click **+ Create.**



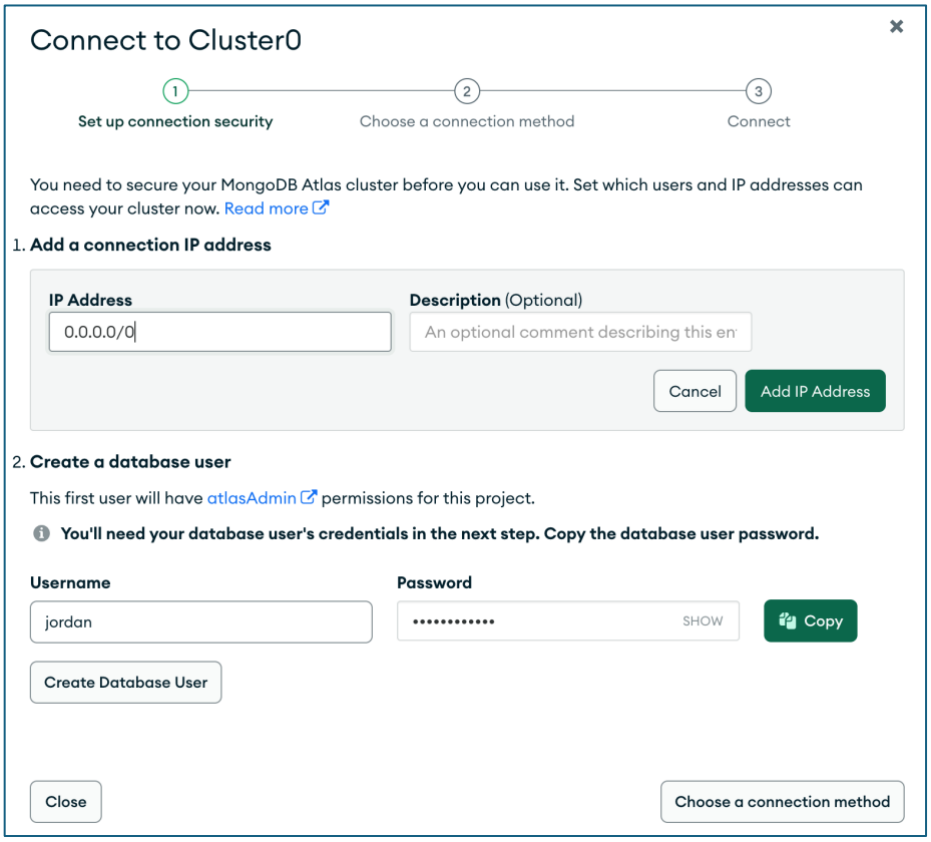
Step 3: Create a Cluster

You'll be presented with some options for your new cluster; choose the **Free** option, uncheck **Automate security setup,** check the box for **Preload sample dataset,** and set the region to **Oregon (us-west-2).** All other settings can be left with the defaults. Click **Create Deployment:**



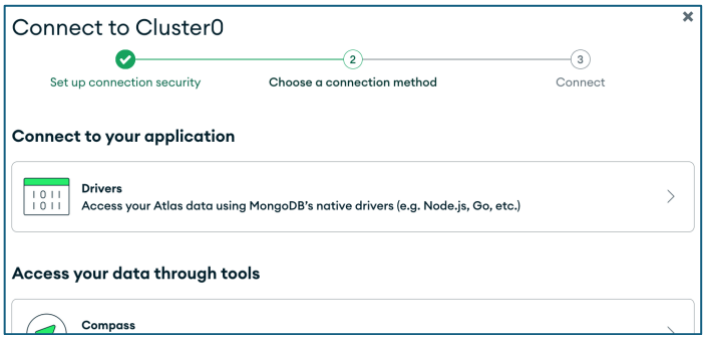
You'll then be presented with a screen like the one shown below. Click **Add a Different IP Address** and enter **0.0.0.0/0,** which will allow connections to your database from anywhere. Click **Add IP Address.** In a real-life production environment, we would never allow connections from anywhere for security reasons, but it's sometimes necessary in development since we'll be connecting to our database from a GitHub Codespace with an unpredictable IP address.

Now choose a username and password; I recommend that you ***avoid special characters in your password and just use lowercase letters and numbers***, as anything more complex than that can lead to encoding issues. Also, note that I and the marker will have access to your password, so don't choose the same password that you use for anything else.



Click **Create Database User** and then **Choose a connection method.**

Of the options presented, choose **Drivers:**



On the next screen, notice the **connection string** but **DO NOT** copy it yet. We won't copy it until we have a safe, secure place to put it. You can access this string later and use it to connect your Node.js/Express application to the database.

Then click **Done.**

Step 4: Set up Credentials for Application

Note: this next part requires some knowledge of Node.js and Express. If you're not enrolled in my WMDD 4936 course and you've never taken it before or have forgotten how to set up an Express application, please contact me for help.

Now create a new GitHub Codespaces project with the blank template, as usual, and initialize a new Node.js project. In the application entry point file (often **app.js)**, initialize a new Express application but don't listen for HTTP requests just yet:

```
import express from "express";
const app = express();


let server;
```

Before we program our application to connect to the database service, we must recognize that we have new security responsibilities, as we'll be using secret credentials to authenticate this connection. ***It is extremely important that you do not commit your credentials to a public GitHub repository.*** *Doing so will result in an immediate security breach and major problems for you. In particular, we must protect the* ***connection string and user password*** *we set up earlier.*

Luckily, we have some simple procedures we can follow to prevent this from happening. First create a file in the root directory of your project called **.gitignore** (note the dot). In this file, add the following, and **save**.

```
node_modules
.env
```

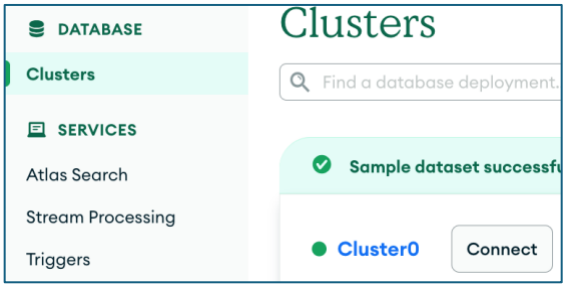
The **.gitignore** file is used by Git to exclude certain files when committing the project. This file instructs Git to NOT commit files or folders with the above names to any repository, private or public. That means we can store secure credentials in the **.env** file.

Now, create a file in the root directory of your project called **.env** (again note the dot). This is a special file used by Node.js to get environment variables, particularly secure credentials such as our database connection string. In this file, add the following:

```
CONNECTION_STRING=
```

At this point, you may want to create a repository and commit the project to confirm that the .env file is NOT included.

You can access your **connection string** again from the MongoDB Atlas dashboard by clicking **Clusters** in the menu on the left, and click **Connect:**



Navigate to **Drivers** once again, and copy the connection string. In **.env,** paste it into the right-hand side of the variable assignment, like this…

```
CONNECTION_STRING=mongodb+srv://jordan:<db_pass....
```

... but of course with your actual full connection string with your own user name. Note that string quotes are NOT used here, as this isn't JavaScript code.

Replace <db_password> with your actual password (and no <angle brackets>). If you've forgotten it, you can reset it from the MongoDB Atlas dashboard by navigating to **Database Access** in the menu bar on the left, and either clicking **EDIT** next to your current user, or creating a new user.

Save the **.env** file.

Step 5: Connect to the Database

Now that we're getting a remote database involved in our application, we need to think about **asynchronous** operations. In particular, to ensure we don't accidentally try to interact with the database before a connection has been established, we must first initiate the database connection, wait for confirmation that the connection was made, and **then** start listening for HTTP requests on our server.

First, install the **MongoDB Node.js driver** in your project:

npm install mongodb

Then create a new file in the root of your project directory called **database.js.** Paste the following code into this file and save:

```
import { MongoClient, ServerApiVersion } from 'mongodb';

// Create a new client to manage connections with the database
// process.env.CONNECTION_STRING is an environment variable with your credentials, pulled from .env
const client = new MongoClient(process.env.CONNECTION_STRING, {
  serverApi: {
    version: ServerApiVersion.v1,
    strict: true,
    deprecationErrors: true,
  }
});
// establish the connection
const connection = client.connect();
const database = client.db('our-first-database');
```

```
// export the connection and client for use in app.js
export { connection, database };
```

As you can see, the above code handles the database connection stuff, creates a new database, and exports the necessary objects. In **app.js**, paste the following code:

```
import { connection, database } from './database.js';


connection.then(()=>{
  console.log("Successful connection to database!");
  server = app.listen(3000, ()=>console.log('Server listening.'));
})
.catch(e=>console.error(e));
```

As you can see, the above code is checking that the connection to the database cluster was successful, and starting the HTTP server if so. (If we can't connect to the database for some reason, we don't even want the HTTP server to run, and we output an error instead.)

You're now free to continue developing the Express application as you normally would, submitting and retrieving documents from the database! (We'll learn how to do this in a future lesson.)

There's only one catch: when starting the application, you must instruct Node which file contains the environment variables. You can do that by including an extra flag when you start the application in the terminal:

node --env-file=.env --watch app.js

If you encounter an error related to MongoDB when running the application this way, try the following:

- Wait 10 minutes. Sometimes it take a while for your credentials to be set up in MongoDB Atlas.
- Make sure you've configured the network settings in MongoDB Atlas to accept connections from anywhere.
- Try creating a new user name and password for the database user in MongoDB. (Note that this is NOT the same as your MongoDB Atlas user account.) Make sure the username and password are correct in the connection string. Don't use uppercase letters or special characters in the password or username.

Now there's only one thing left to do: make sure our application closes the database connection when the application is manually terminated through a Ctrl+C command in the terminal. Paste this code at the bottom of database.js:

```
process.on('SIGINT', ()=>{
    client.close();
    console.log("closed database connection");
    process.exit(1);
})
```

The 'SIGINT' code is used to catch a manual shutdown event when it occurs, and the following function, which closes the connection, will be called. The application is then terminated by calling **process.exit().**

That's it for now. In a future lesson, you'll learn how to actually send and retrieve data with the database.