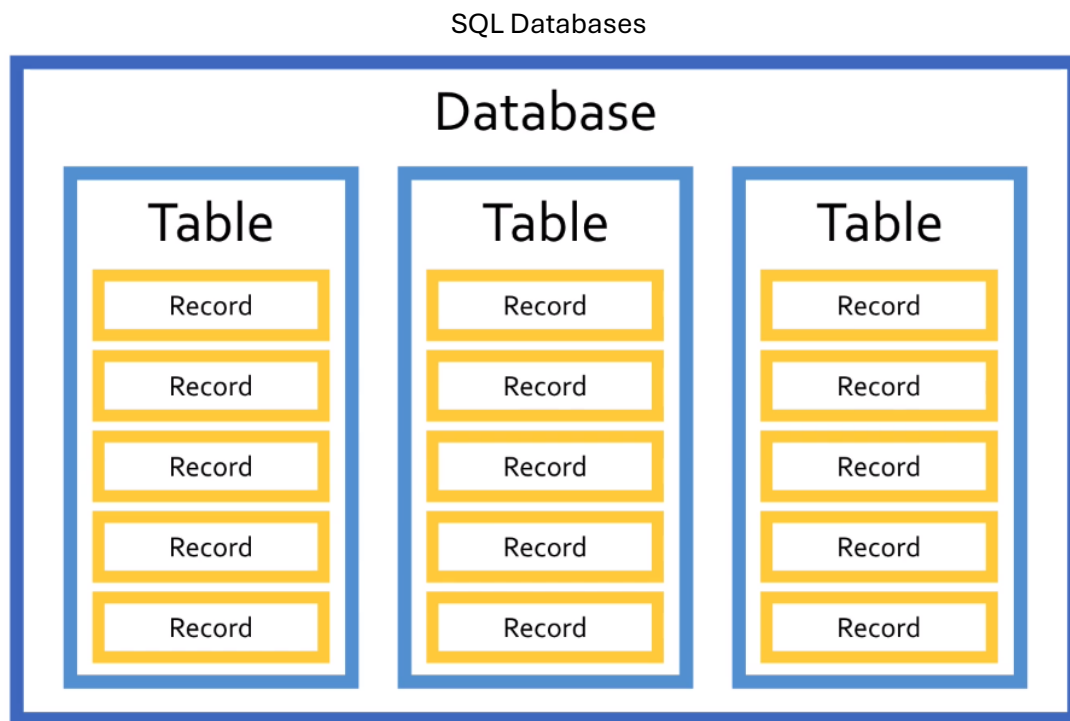


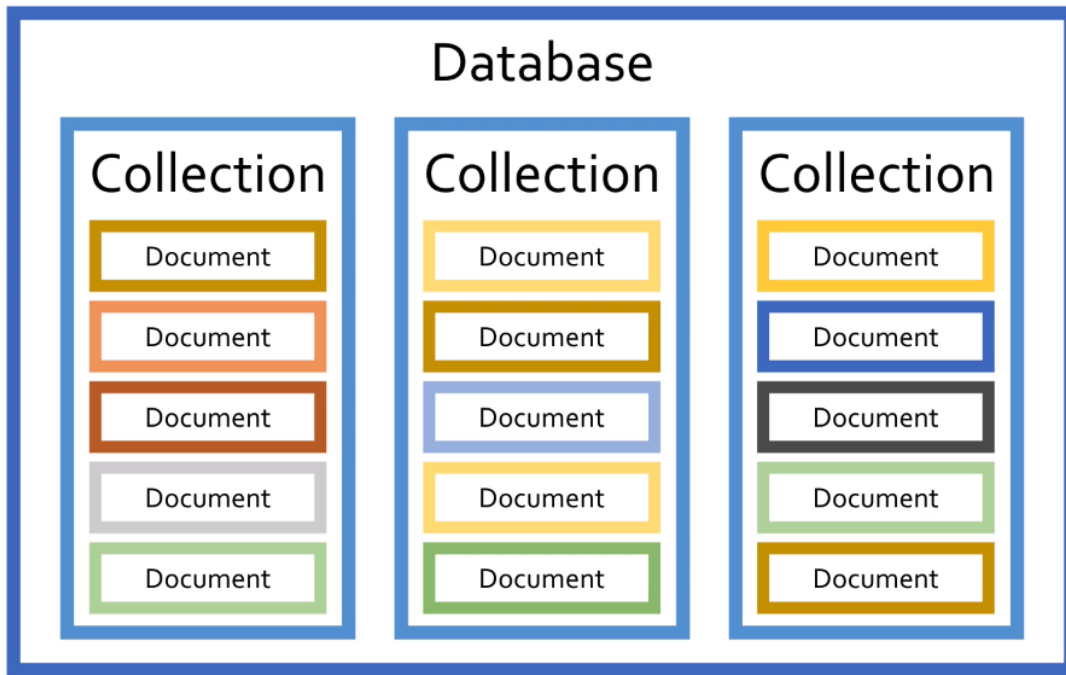
NoSQL Databases

SQL vs NoSQL

Schematic layout of differences between SQL (relational) and NoSQL databases are illustrated as follows:



NoSQL databases:



Database Concepts

Tabular (Relational)	MongoDB
Database	Database
Table	Collection
Row	Document
Index	Index
Join	\$lookup
Foreign Key	Reference

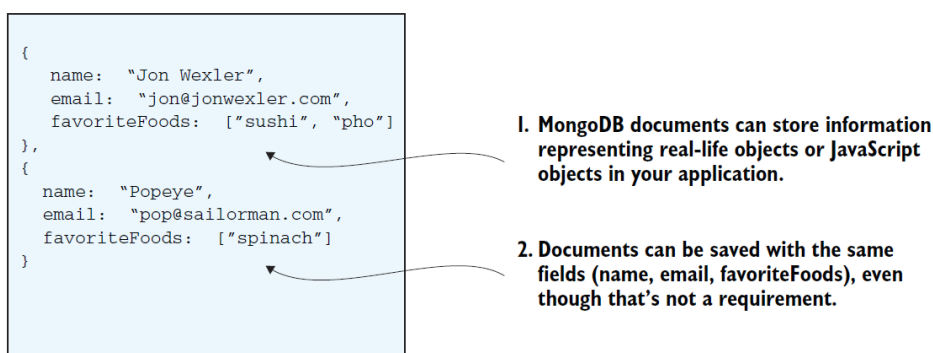
MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. It is the most popular database for modern apps. **MongoDB** Atlas is the global cloud database on AWS, Azure, and GCP.

In the following, we will discuss MongoDB database and explore working with MongoDB in several simple web applications with the goal of store and retrieve data for our projects.

SETTING UP A MONGODB DATABASE

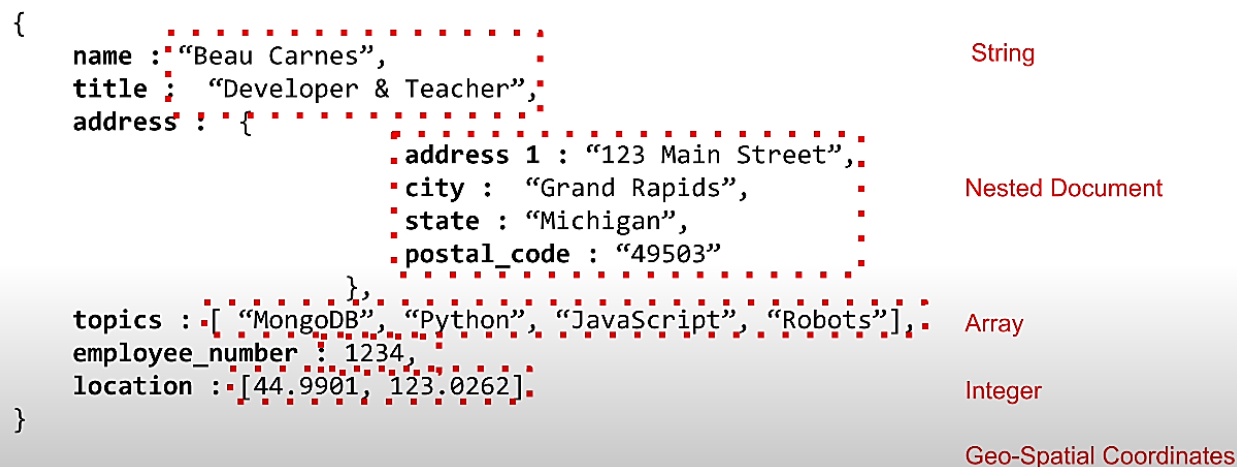
Storing data is the most important part of application development. Without long-term storage, you're limited in the way you can interact with your users. The data in every application you've built to this point disappeared each time you restarted the application. A database is an organization of your data designed for easy access and efficient changes made by your application that you can store your data in. Like a web server, your application connects to a MongoDB database and requests data.

MongoDB is an open-source database program that organizes data by using documents. MongoDB documents store data in a JSON-like structure, allowing you to use key-value pairing to associate data objects with properties. This system of storage follows a familiar JavaScript syntax. Notice in the following figure that a document's contents resemble JSON.



In fact, MongoDB stores documents as BSON (a binary form of JSON).

MongoDB Documents – BSON Types



You have likely learned about relational databases already. MongoDB is not relational database and is used differently. However, MongoDB's nonrelational database system leads the Node.js application community. Similar approaches as to how to work with relational databases may be applied to MongoDB. MongoDB also has an ORM that is called Mongoose. The related techniques

and tools are summarized in this document to help you to understand the basics of setting up and working with MongoDB.

You could set up a relational database with Node.js—in fact, many applications do—but to best make use of a SQL database, it helps to know how to write in the SQL language. The MongoDB query language is simpler to understand for people who have a Java-Script background.

Step 1 - Install MongoDB

For installation of MongoDB on Windows, go to this link (if you use Mac you may need to install MongoDB using Homebrew using “brew install mongodb”):

<https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-windows/>

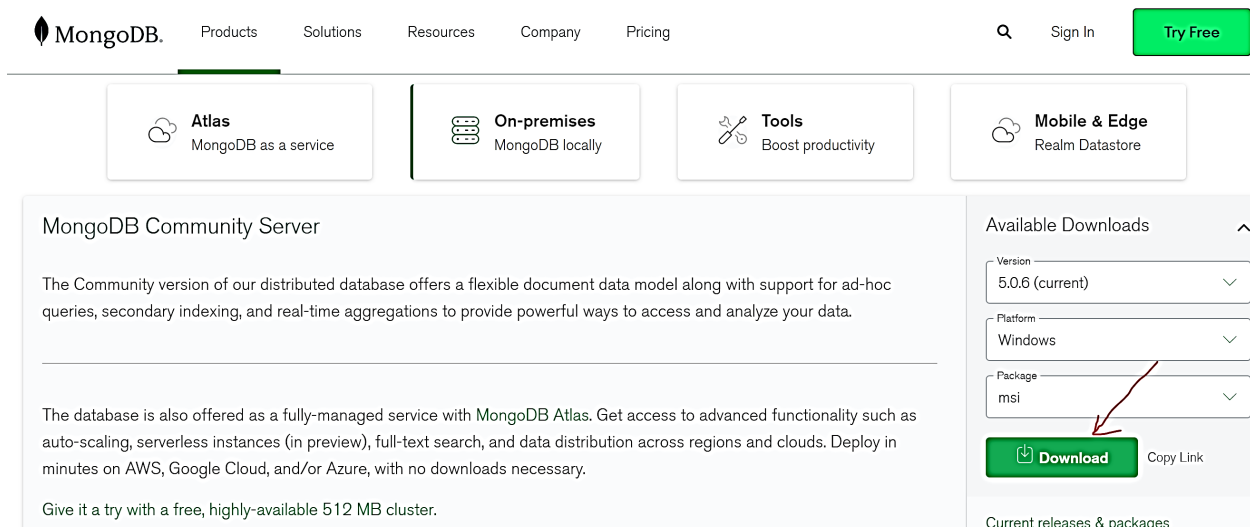
<https://www.mongodb.com/try/download/community>

we'll work on setting up and install **MongoDB** so we can use it in future projects.

Installation

First, you need to **download MongoDB Community Server** from

https://www.mongodb.com/try/download/community?tck=docs_server&_ga=2.75162203.838359985.1646123076-2036510034.1646123076



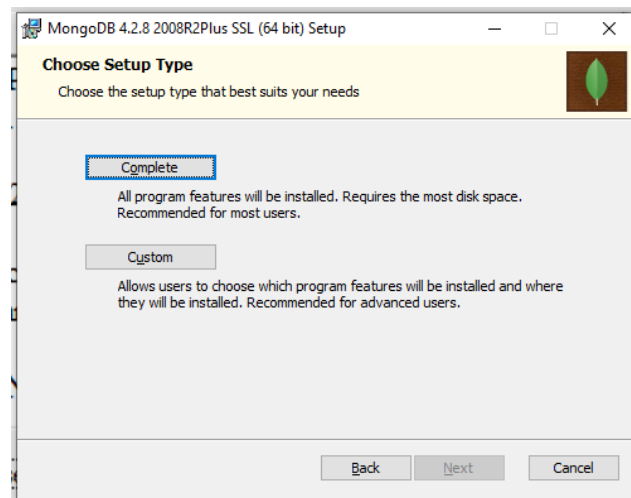
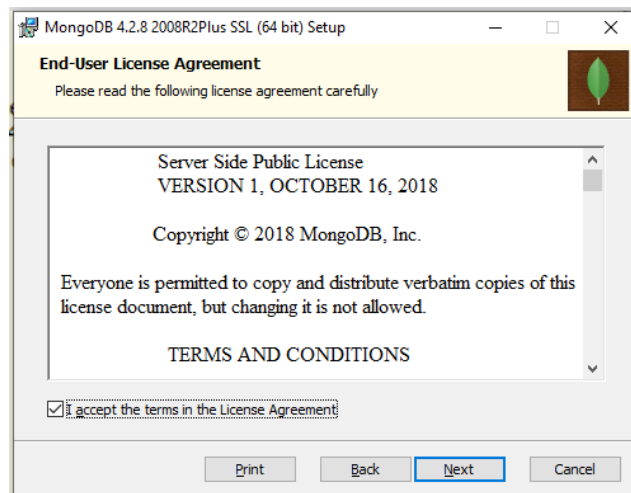
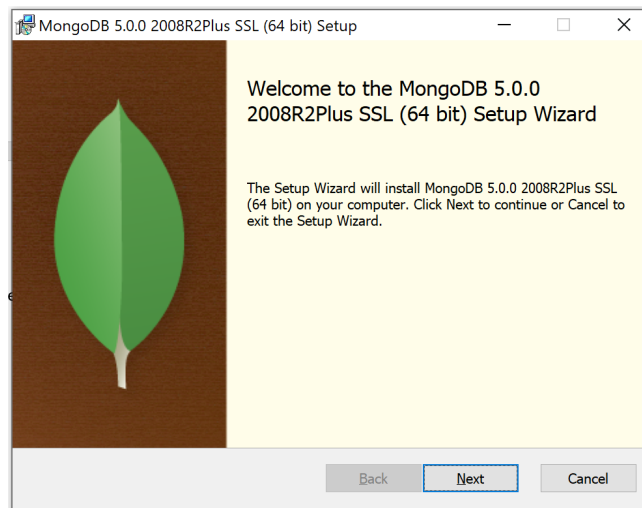
The screenshot shows the MongoDB website's download page. The navigation bar includes links for Products, Solutions, Resources, Company, and Pricing, along with a search icon, a Sign In button, and a Try Free button. Below the navigation bar, there are four product categories: Atlas (MongoDB as a service), On-premises (MongoDB locally), Tools (Boost productivity), and Mobile & Edge (Realm Datastore). The main content area is titled 'MongoDB Community Server' and describes the community version of the database. It also mentions that the database is offered as a fully-managed service with MongoDB Atlas. On the right side, there is a section titled 'Available Downloads' with dropdown menus for Version (5.0.6 (current)), Platform (Windows), and Package (msi). A red arrow points to the 'Download' button in this section. Below the download button is a 'Copy Link' button. At the bottom of the section, it says 'Current releases & packages'.

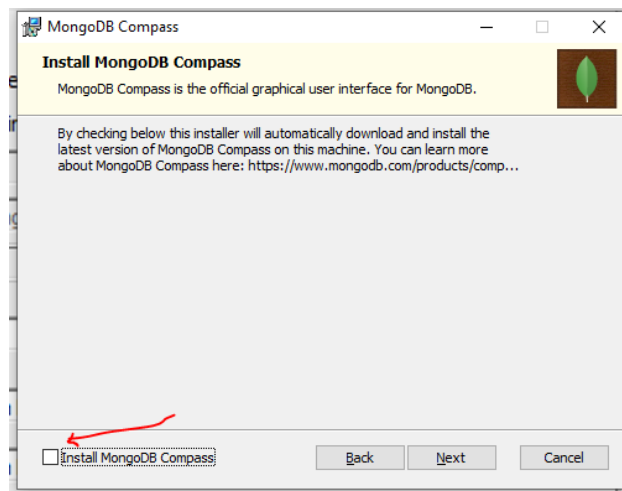
For installation of MongoDB on Windows, go to this link (if you use Mac you may need to install MongoDB using Homebrew using “brew install mongodb”):

<https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-windows/>

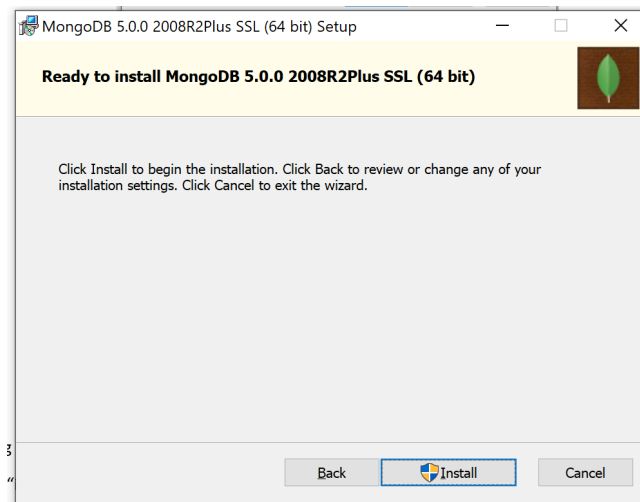
<https://www.mongodb.com/try/download/community>

follow the following steps in each picture below:



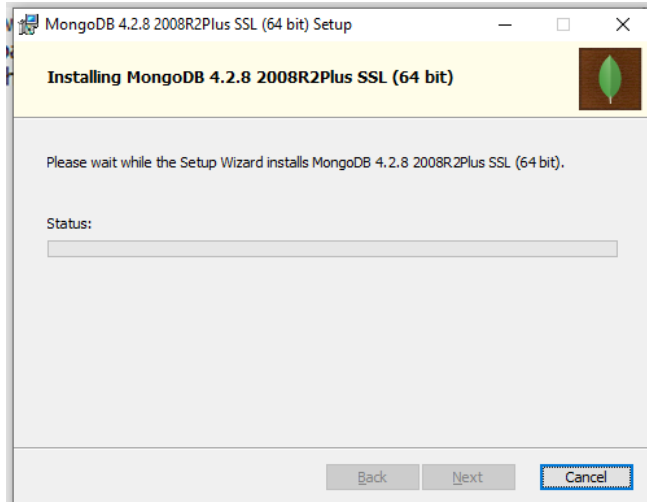


Installing Compass provides a GUI to work with the database, and is optional.



Click on “install”

Also you may want to install MongoDB tools too



NOTE In Windows, you may need to add the MongoDB folder path to your environment's PATH variable. Under "System variables", click on Path then on "Edit". On the new window that pops up, click on "New" and paste your MongoDB Server directory path (it should look like

"C:\Program Files\MongoDB\Server\5.0\bin",

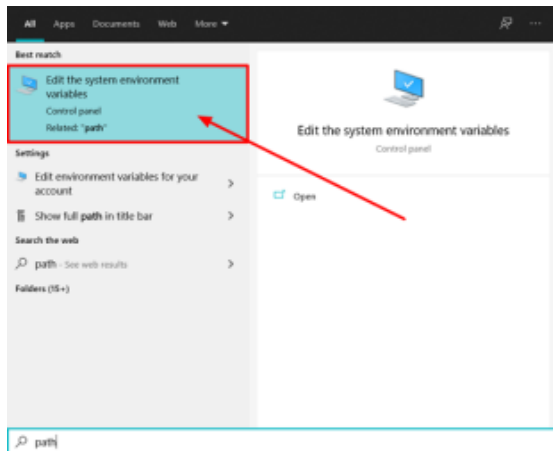
just mind the version you're using in this example 5.0). Click on "New" again to add the path for the MongoDB Tools as well ("C:\Program Files\MongoDB\Tools\100\bin"). Hit "Ok" and you're ready to move on to the next lesson.

MongoDB Command Line Database Tools Download

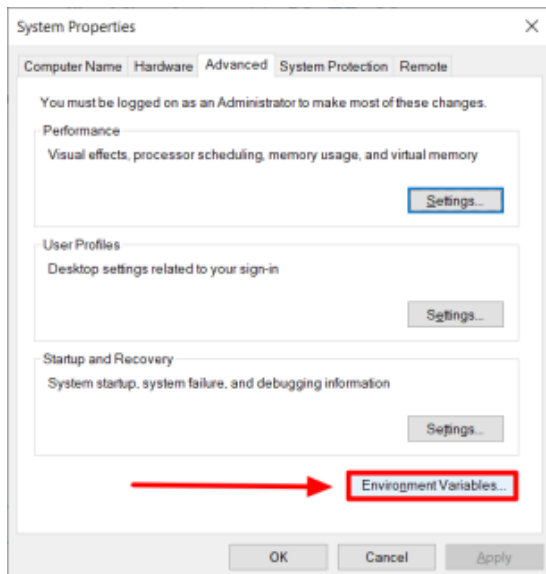
MongoDB Shell	▼
MongoDB Compass	▼
MongoDB CLI for Cloud	▼
MongoDB Database Tools <p>The MongoDB Database Tools are a collection of command-line utilities for working with a MongoDB deployment. These tools release independently from the MongoDB Server schedule enabling you to receive more frequent updates and leverage new features as soon as they are available. See the MongoDB Database Tools documentation for more information.</p>	Available Downloads ^ <div> Version 100.5.2 </div> <div> Platform Windows x86_64 </div> <div> Package zip </div> <div> Download Copy Link </div> <div> Documentation Archived releases </div>

Adding MongoDB to the System's Path Variable

Now, you may need to **add** this same **MongoDB** directory **path** above to the Window's **PATH variable**. To so, **search** in the Windows search bar for “**path**” and **click** on the following option:



Go to “**Environment variables**”:



Next, under “**System variables**”, click on **Path** then on “**Edit**”.

On the **new window** that pops up, click on “**New**” and **paste** your **MongoDB Server** directory path (it should look like “C:\Program Files\MongoDB\Server\5.0\bin”, just mind the version you’re using in this example 5.0). Click on “**New**” again to add the path for the **MongoDB Tools** as well (“C:\Program Files\MongoDB\Tools\100\bin”). Hit “**Ok**” and you’re ready to move on to the next lesson.

You likely need to manually create the directory by running the command below.

```
mkdir -p /data/db
```

that basically means you need to have a data/db folder in the root directory (C:/). Without it some installations may fail.

Create a folder called db within another folder called data at your computer's root level (as far back as you can cd .. in a terminal window). You can create this folder by entering **mkdir -p /data/db** in a terminal window.

You may need to give permissions to your user account to use this folder. To do so, the steps are as follows:

- Go to <https://www.mongodb.com/download-center#community> in your browser.
- Download MongoDB for Windows (.msi).
- When the download is complete, open the file, and click through the default installation steps.
- When the installer completes, go to your C:\ drive, and create a new folder called data and a folder within it called db.

In this lesson, we're going to create a **database** and records using the **Mongo CLI (Command Line Interface)**. However, keep in mind the **CLI** is very powerful and not the usual method for interacting with the database in practice. Rather, a web interface is more common since it acts as a safety measure against accidental commands.

You can test whether Mongo was installed successfully by typing `mongo` in a new terminal window. This command brings up the MongoDB shell, an environment within which you can run MongoDB commands and view data. This shell environment is similar to REPL because it isolates your terminal window to allow you to interact purely with MongoDB syntax. When you have some data to work with, you can further explore this environment.

Step 2 - Running commands in the MongoDB shell

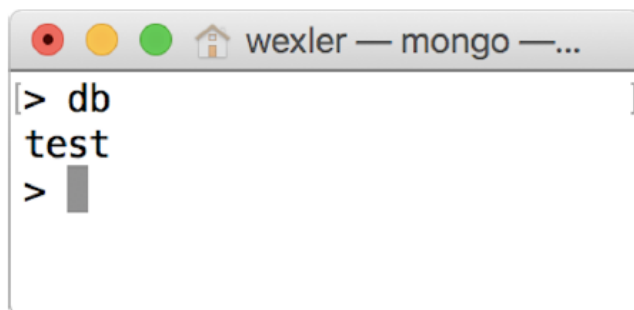
You can test whether Mongo was installed successfully by typing "**mongo**" in a new terminal window. This command brings up the MongoDB shell, an environment within which you can run MongoDB commands and view data. This shell environment is similar to REPL because it isolates your terminal window to allow you to interact purely with MongoDB syntax. When you have some data to work with, you can further explore this environment.

Now that MongoDB is running, it's ready to receive commands to add, view, delete, or otherwise change data. Before you connect MongoDB to your application, you can test some commands in the MongoDB shell.

WARNING Commands that you run in the MongoDB shell are permanent. If you delete data (or an entire database), there's no going back.

Run `mongo` in a new terminal window. This command should prompt the shell to start. You'll be greeted by your MongoDB version number, potentially a few warnings (which you can ignore for now), and the familiar "**>**" to indicate that the shell is active and ready for commands.

MongoDB can store multiple databases; it's a management system for all your applications' databases. To start, the MongoDB shell places you in the test database. You can see this test database by entering `db`, to list your current database, after the prompt.



MongoDB Shell

> show dbs

Creating a new database (>use);

You can create a new database and simultaneously switch into it by entering **use <new_name>**. Try switching to a new database for storing some recipes by entering **"use recipe_db"**. Then run **db** again to see that you're within the **recipe_db** database.

NOTE You won't see your new database in the list of databases until data is added.

Step 3 - Reading and entering data within the MongoDB shell

To add data to your database, you need to specify a collection name with which that data is associated. A MongoDB collection is representative of your data model, storing all documents related to that model within the same grouping. If you want to create a contact list for the recipe application, for example, create a new collection (**contacts**) and add (.insert) a data item with the command shown in the following listing. The insert method runs on a MongoDB collection to add elements of a JavaScript object to a new document.

>

```
db.contacts.insert({
  name: "Jon Wexler",
  email: "jon@jonwexler.com",
  note: "Decent guy."
})
```

Insert new data into the database.

At this point, there's no strict collection structure; you can add any values to new documents without needing to follow previous data patterns.

Exercise: Insert another item into the contacts collection with these properties:

```
{first_name: "Jon",
 favoriteSeason: "spring",
 countries_visited: 42}.
```

MongoDB lets you add these seemingly conflicting data elements.

To list the collection's contents, you can enter

```
> db.conntacts.find ( ).
```

You should see a response that looks like the next listing. Both inserted items are present, with an extra property added by MongoDB. The id property stores a unique value that you can use to differentiate and locate specific items in your database.

```
{ "_id": ObjectId("5941fce5cda203f026856a5d"), "name": "Jon  
Wexler", "email": "jon@jonwexler.com", "note":  
  
  "Nice guy." }  
{ "_id": ObjectId("5941fe7acda203f026856a5e"), "first_name":  
  "Jon", "favoriteSeason": "spring", "countries_visited": 42 }
```

Display results of
database documents.

Try searching for a specific item in the contacts collection by entering

```
> db.contacts.find({_id: ObjectId("5941fce5cda203f026856a5d")})
```

Or

```
> db.contacts.find({name: "Jon Wexler"}).
```

Some useful MongoDB commands are listed as follows:

MongoDB Shell Commands

Command	Description
show collections	Displays all the collections in your database. Later, these collections should match your models.
db.contacts.findOne	Returns a single item from your database at random or a single item matching the criteria passed in as a parameter, which could look like findOne({name: 'Jon'}).
db.contacts.update({name: "Jon"}, {name: "Jon Wexler"})	Updates any matching documents with the second parameter's property values.
db.contacts.delete({name: "Jon Wexler"})	Removes any matching documents in the collection.
db.contacts.deleteMany({})	Removes all the documents in that collection. These commands can't be undone.

For more practice, view the command cheat sheet at <https://docs.mongodb.com/manual/reference/mongo-shell/>.

For Example let's review how we can update data

Updating Data

To begin, let's connect to our database and take a look at the countries collection.

use recipe_db

show collections

db.contacts.find()

As you might imagine, the email might change in contacts, so we need the ability to update them to keep our database current. The function for updating is simply **update**. Within this function, we need to pass in two filters. The first filter we need to pass in is a regular query filter that will grab the record we want to update. Then, the second filter we pass in is the update.

For our example, let's theoretically say the email of John Wexler changes to jonjwexler@gmail.com. To query for the record, we will use the "name" we've been setting ("jon wexler" in this case). For our update filter, we need to use a special Mongo keyword again called set. With set, it will set the email key

only within our queried record to the value we specify. Note that set requires a JSON object to contain the entire update.

db.contacts.update({name:"jon wexler"}, {\$set:{email:"jonjwexler@gmail.com"}})

you can now check and see the record has been updated.

If you'd like to learn more about updating in MongoDB, please check out the relevant documentation:

<https://docs.mongodb.com/manual/tutorial/update-documents/>

for deleting you can use deleteOne or deleteMany. Examples are provided below:

db.contacts.deleteOne({email:"a@a.com"})

db.countries.deleteMany({Note:"NA"})

Now we have the basic CRUD functions of MongoDB down. In the next few lessons, we're going to create a small project using an ORM (Object Relational Mapper) called Mongoose.

For more information on deleting data in MongoDB, check out the documentation on the subject: <https://docs.mongodb.com/manual/tutorial/remove-documents/>

you can insert a file into a MongoDB database with **db.collection.insert(file)**

the exercise below shows inserting a file

Exercise:

Use the following dataset: (let's name it mylibrary and name the collection bookInfo)

- put the following dataset into a file, name it mybooks.json

```
[  
  { "book": "Civilization", "author": "Tzu", "year": "1922" },  
  { "book": "Urbanization", "author": "Richards", "year": "1955" },  
  { "book": "Concuring the World", "author": "Alexander", "year": "1022" }  
]
```

```
var file = cat('./mybooks.json')    # file name  
use mylibrary                      # db name  
var libjson = JSON.parse(file)     # convert string to JSON  
db.forms.insert(libjson)           # collection name
```

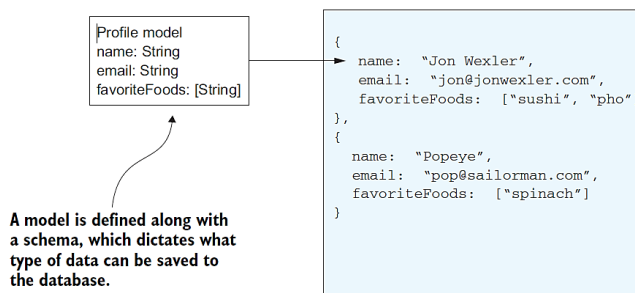
- Create the dataset, and collection
- type all data to the shell terminal
- type rows that have "grade": "A"

Step 4 - Connecting MongoDB to a Node.js application

About Mongoose

Mongoose is an object-document mapper (ODM) that allows you to run MongoDB commands in a way that preserves the object-oriented structure of your application. Mongoose offers tools to build

models with schemas defining what type of data can be saved. A model is like a class for a JavaScript object that Mongoose uses to organize your database queries.



To use Mongoose, you need to create a node.js app to perform the following instructions.

Setting up Mongoose with your Node.js application

We can install Mongoose and see what a model looks like in an application.

To install Mongoose enter the following in NPM:

```
$ npm install mongoose
```

You also need to install other dependencies including Express.

The first things we want to do are import express and mongoose. We also want to create an instance of express.

```
const express = require('express')
```

```
const mongoose = require('mongoose')
```

```
const app = express()
```

With this set up, we will also create a simple path for the home page. This simple function will take in the usual request, response, and next arguments needed for this sort of function. For our response, we will return a JSON object that has a confirmation and data key.

```

app.use('/', (req, res, next) => {
  res.json({
    confirmation: 'success',
    data: 'This is the Mongo project!' })
})
  
```

So that we can test this page, we need to tell our app to listen to port 3000. We will also log to the console whenever our server is running.

```
app.listen(3000)

console.log('App running http://localhost:3000')
```

Now we have our most basic project set up. In the next lesson, we will connect Mongoose to our MongoDB database and set up some of the tools we'll be using to query against the database.

The **connection** code for **Mongoose** is as simple as using the **.connect** command. With this, we are returned a **promise**. This means that within our **connection**, we need to provide a block that tells our code what to do when it succeeds (**.then**) and what to do when it fails (**.catch**). No matter what happens, one of these code blocks will run.

```
// connect to our Mongo DB:

mongoose.connect()

.then(data => {
})

.catch(err => {
})
```

Add the following code in the following listing to main.js. Require mongoose into the application file. Set up the application's connection to your MongoDB database. Then assign the database connection to the db variable, which you can use later in the file for data changes or database state changes.

Lastly, we want to add console logs to our then and catch blocks. For the catch block, we will log the actual error as a message within the statement so we can see exactly what went wrong.

```
// connect to our Mongo DB:

mongoose.connect('mongodb://localhost:27017/recipe_db', { useNewUrlParser: true })

.then(data => {
  console.log('Mongo DB connection success!')
})

.catch(err => {
  console.log('Mongo DB connection failed: ' + err.message)
})
```

That's all you need to do to set up Mongoose. You can log a message as soon as the database is connected by adding the code in the next listing to `main.js`. The database connection runs the code in the callback function (the log message) only once upon receiving an "open" event from the database.

Step 5 - Creating a schema

A schema is like a class definition in some languages or, more broadly, a blueprint for how you want data to be organized for specific objects in your application. To avoid inconsistent data, where some documents have an email field and others don't, for example, you can create a schema stating that all contact objects need to have an email field to get saved to the database.

As mentioned, MongoDB itself is a schema-less database. However, Mongoose enforces schemas on our behalf to help maintain data integrity. If we enter a record incorrectly, Mongoose will reject the record and tell us why. As the developer, though, it's on us to set up the schema in the first place.

Let's say you want to add a newsletter subscription form to an application, create a schema for the subscriber. Add the code from the following listing to your `node.js` code (let's say in `main.js`). We can use Mongoose that is MongoDB ORM.

mongoose.Schema offers a constructor that allows you to build a schema object with the given parameters. Then add object properties to state the name of the object's field and its data type. Someone's name can't be a number, for example.

Subscriber schema in `main.js`

```
const subscriberSchema = new mongoose.Schema({
  name: {type: String, default: ""},
  email: {type: String, default: ""},
  zipCode: {type: Number, default: 0}
})
```

Or you can summarize as the code below:

```
const subscriberSchema = mongoose.Schema({
  name: String,
  email: String,
  zipCode: Number
});
```

← Create a new schema with `mongoose.Schema`.

← Add schema properties.

NOTE MongoDB isn't enforcing your schema; Mongoose is. For more information about Mongoose schema data types, visit <http://mongoosejs.com/docs/schematypes.html>.

Now that the schema is defined, you need to apply it to a model by using

const Subscriber = mongoose.model("Subscriber", subscriberSchema).

The model is what you'll use to instantiate new Subscriber objects, and the schema you created can be used for that model. The model method takes a model name of your choosing and a previously defined schema (in this case, the subscriberSchema).

You can instantiate new objects from this model by referring to Subscriber. You have two ways to generate new objects, as shown in the following listing. You can construct a new instance of the Subscriber model by using the new keyword and by passing properties that abide by the subscriberSchema earlier in the section. To get this newly created Subscriber object into the database, you can call save on it and handle any errors or returned data through a callback function.

An error may have to do with data that doesn't match the schema types you defined earlier. The saved item returns data that you can use elsewhere in the application. You may want to thank the subscriber by name for signing up, for example. create does what new and save do in one step. If you know that you want to create and save the object right away, use this Mongoose method.

```

var subscriber1 = new Subscriber({
  name: "Jon Wexler",
  email: "jon@jonwexler.com"
});

subscriber1.save((error, savedDocument) => {
  if (error) console.log(error);
  console.log(savedDocument);
});

Subscriber.create(
  {
    name: "Jon Wexler",
    email: "jon@jonwexler.com"
  },
  function (error, savedDocument) {
    if (error) console.log(error);
    console.log(savedDocument);
  }
);

```

Instantiate a new subscriber.

Save a subscriber to the database.

Log saved data document.

Pass potential errors to the next middleware function.

Create and save a subscriber in a single step.

Add the code from the listings in this section to your main.js file. As soon as you start the application with node main.js, you should see your MongoDB recipe_db database populate with a new subscriber.

mongoDB shell commands:

- select a db: **use choose_db**
- show all dbs: **show dbs**
- show collections in a db: **use choose_db** then **show collections**

- show all records in a db: **use choose_db** then **db.users.find()**
- delete a db: **use choose_db** then **db.dropDatabase()**
- to see which db is being used: **db**

Working with mongodb on the “usersDB” database

Create a new folder **trymongoose**

- npm init -y
- npm install mongoose
- create migration.js

```
const express = require('express')
const mongoose = require('mongoose')
const app = express()

mongoose.connect("mongodb://localhost:27017/usersDB2", {
  useNewUrlParser: true,
});

//creating schema
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  fav_pizza: String,
  space_invaders: Number,
});

// defining model
const User = mongoose.model("User", userSchema);

//User has to be with capital first letter. mongo will make that small and pluralize
// the model name: "User" => "users"
```

```
// creating
const pj = new User({
  name: "PJ",
  email: "pj@company.org",
  fav_pizza: "Pepperoni",
  space_invaders: 826488,
});

const trish = new User({
  name: "Trish",
  email: "trish@company.org",
  fav_pizza: "Spicy Veg",
  space_invaders: 674588,
});

const paddy = new User({
  name: "Paddy",
  email: "paddy@company.org",
  fav_pizza: "Ham",
  space_invaders: 998988,
});

const bob = new User({
  name: "Bob",
  email: "bob@company.org",
  fav_pizza: "Onion",
  space_invaders: 657745,
});

const alice = new User({
  name: "Alice",
  email: "alice@company.org",
  fav_pizza: "Everything",
```

```

    space_invaders: 929848,
  });

  User.insertMany([pj, trish, paddy, bob, alice], function (err) {
    if (err) {
      console.log(err);
    } else {
      console.log("successfully created db");
    }
  });

  app.listen(3000)
  console.log('App running http://localhost:3000')

```

main.js

You can use MongoDB terminal to check the new databases.

Read Data

To read data explore the following code:

```

const mongoose = require('mongoose')

//create and/or connect to a db
mongoose.connect("mongodb://localhost:27017/usersDB2", {
  useNewUrlParser: true,
});

//creating schema
const userSchema = new mongoose.Schema({
  name: String,
  email: String,

```

```
    fav_pizza: String,
    space_invaders: Number,
  });

// defining model
const User = mongoose.model("User", userSchema);

User.find(function (err, users) {
  if (err) {
    console.log(err);
  } else {

    // console.log(users);
    // if we want to show only the names the following code does
    users.forEach(function(user){
      console.log(user.name)

      // closing the connection: good practice
      mongoose.connection.close();

    } )

  }

});
```

Read.js

Update Data

To update check the following code:

```
const mongoose = require('mongoose');

//create and/or connect to a db
mongoose.connect('mongodb://localhost:27017/usersDB2', {
  useNewUrlParser: true,
});

//creating schema
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  fav_pizza: String,
  space_invaders: Number,
});

// defining model
const User = mongoose.model('User', userSchema);

User.updateOne({_id: '5f1681eeb0129c2b24eb7fc4'}, {name:'Alice Updated'}, function(err){
  if (err) {
    console.log(err);
  } else {
    console.log('marking')
  }
});

User.find(function (err, users) {
  if (err) {
    console.log(err);
  }
});
```

```

    } else {

        // console.log(users);
        // if we want to show only the names the following code does
        users.forEach(function(fruit){
            console.log(fruit.name)

            // closing the connection: good practice
            mongoose.connection.close();

        } )
    }
});

```

Update.js

Deleting Data

Create delet.js and check the code

```

const mongoose = require("mongoose");

//create and/or connect to a db
mongoose.connect("mongodb://localhost:27017/usersDB2", {
    useNewUrlParser: true,
});

//creating schema
const userSchema = new mongoose.Schema({
    name: String,
    email: String,
    fav_pizza: String,
    space_invaders: Number,
});

// defining model

```

```
const User = mongoose.model("User", userSchema);

User.deleteOne({_id: "5f1681eeb0129c2b24eb7fc4"}, {name:"Alice Updated"}, function(err){

    if (err) {

        console.log(err);

    } else {

        console.log('deleted')

        // console.log(users);

        // if we want to show only the names the following code does

    }

});

User.find(function (err, users) {

    if (err) {

        console.log(err);

    } else {

        // console.log(users);

        // if we want to show only the names the following code does

        users.forEach(function(fruit){

            console.log(fruit.name)

            // closing the connection: good practice

            mongoose.connection.close();

        } )

    }

});
```


Next step is to discuss using MongoDB in the cloud using a service called **MongoDB Atlas**.

MongoDB Atlas:

Getting Started With MongoDB Atlas




- The fastest, easiest way to get started with MongoDB.
- <https://www.mongodb.com/cloud/atlas>

For MongoDB Atlas, first create an account and sign in




Get started free

No credit card required

 Sign up with Google

or

Your Company (optional)

How are you using MongoDB ? 

Your Work Email

First Name

Last Name

Password

Myself Access Manager Support Billing

Project2 Atlas Realm Charts

DATA STORAGE

Clusters

Triggers

Data Lake

SECURITY

Database Access

Network Access

Advanced

MYSELF > PROJECT2

Clusters

Find a cluster...

Create a cluster

Choose your cloud provider, region, and specs.

[Build a Cluster](#)

Once your cluster is up and running, live migrate an existing MongoDB database into Atlas with our [Live Migration Service](#).

MONGODB ATLAS

Choose a path. Adjust anytime.

Available as a fully managed service across 60+ regions on AWS, Azure, and Google Cloud

Shared Clusters

For teams learning MongoDB or developing small applications.

- ✓ Highly available auto-healing cluster
- ✓ End-to-end encryption
- ✓ Role-based access control

[Create a cluster](#)

Starting at
FREE

Dedicated Clusters

For teams building applications that need advanced development and production-ready environments.

- ✓ Includes all features from Shared Clusters
- ✓ Auto-scaling
- ✓ Network isolation
- ✓ Realtime performance metrics

[Create a cluster](#)

Starting at
\$0.08/hr*
*estimated cost \$56.94/month

Dedicated Multi-Region Clusters

For teams developing world-class applications that require multi-region resiliency or ultra-low latency.

- ✓ Includes all features from Shared and Dedicated Clusters
- ✓ Replicate data across multiple regions
- ✓ Globally distributed read and write operations
- ✓ Control data residency at the document level

[Create a cluster](#)

Starting at
\$0.13/hr*
*estimated cost \$98.55/month

Create a cluster

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our [documentation](#).

Cloud Provider & Region

AWS, N. Virginia (us-east-1) ▾

aws

Google Cloud Platform

Azure

★ Recommended region ⓘ

NORTH AMERICA

EUROPE

ASIA

🇺🇸 N. Virginia (us-east-1) ★

🇺🇸 Oregon (us-west-2) ★

🇩🇪 Frankfurt (eu-central-1) ★

🇩🇪 Ireland (eu-west-1) ★

🇮🇳 Mumbai (ap-south-1)

🇸🇬 Singapore (ap-southeast-1) ★

AUSTRALIA

🇦🇺 Sydney (ap-southeast-2) ★

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage) >
Encrypted

Additional Settings

MongoDB 4.2, No Backup >

FREE

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Back

Create Cluster

🇺🇸 N. Virginia (us-east-1) ★

🇩🇪 Frankfurt (eu-central-1) ★

🇸🇬 Singapore (ap-southeast-1) ★

🇺🇸 Oregon (us-west-2) ★

🇩🇪 Ireland (eu-west-1) ★

🇮🇳 Mumbai (ap-south-1)

AUSTRALIA

🇦🇺 Sydney (ap-southeast-2) ★

Cluster Tier

M0 Sandbox (Shared RAM, 512 MB Storage) >
Encrypted

Additional Settings

MongoDB 4.4, No Backup >

Cluster Name

nodeexpress-jwt-test ▾

One time only: once your cluster is created, you won't be able to change its name.

nodeexpress-jwt-test

Cluster names can only contain ASCII letters, numbers, and hyphens.

FREE

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Back

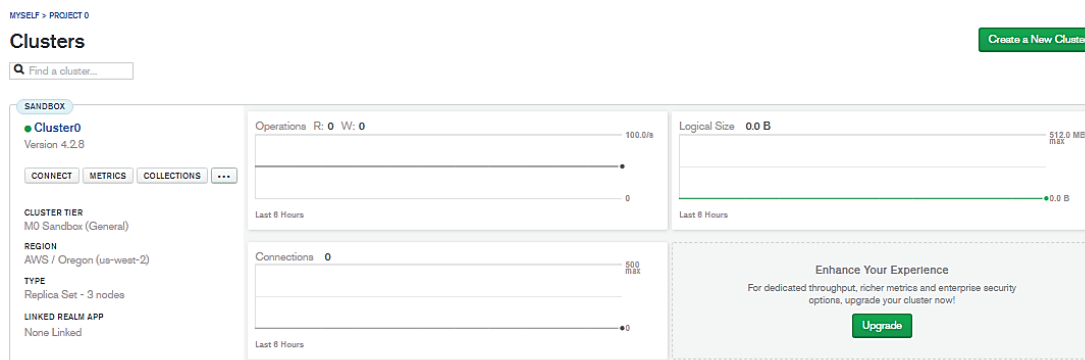
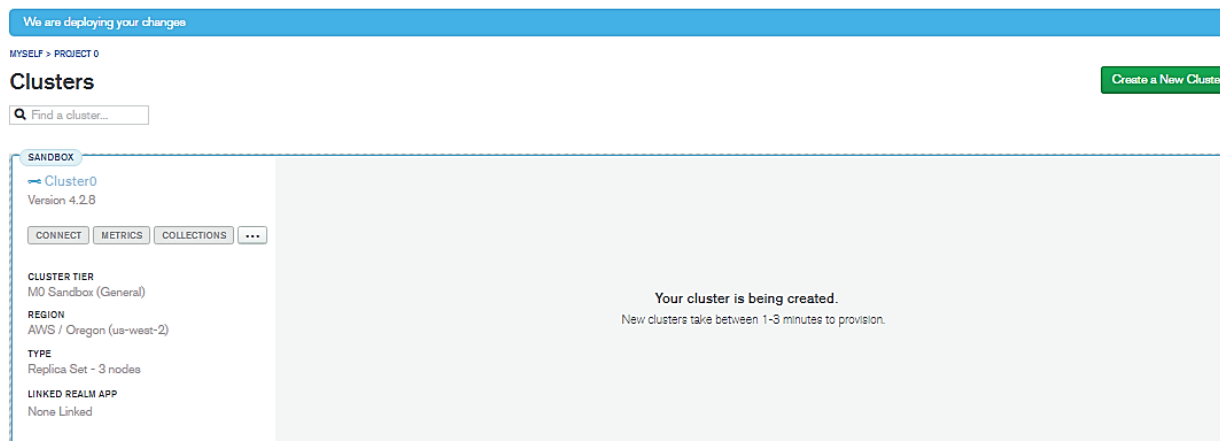
Create Cluster

It takes a few minutes to create the cluster

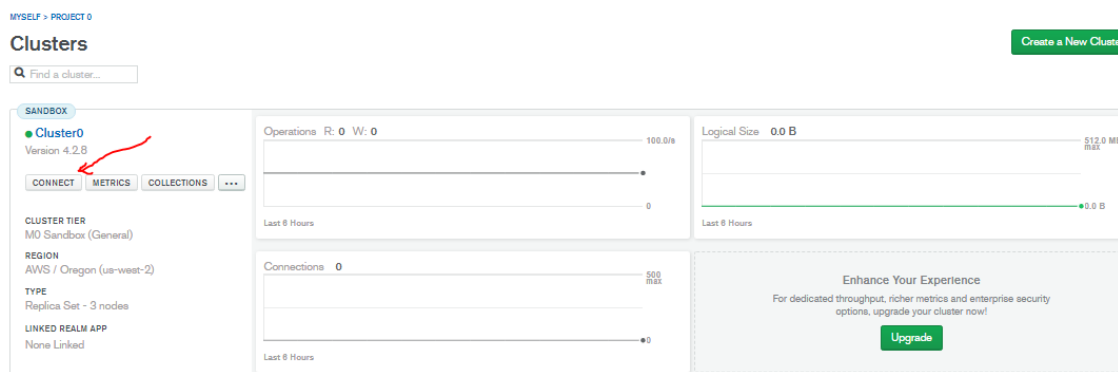
After the cluster is created, you will have to configure your security. The two things we are required to setup from a security standpoint are

- IP Whitelist addresses and
- a database user.

For the IP Whitelist, just add your current IP address.

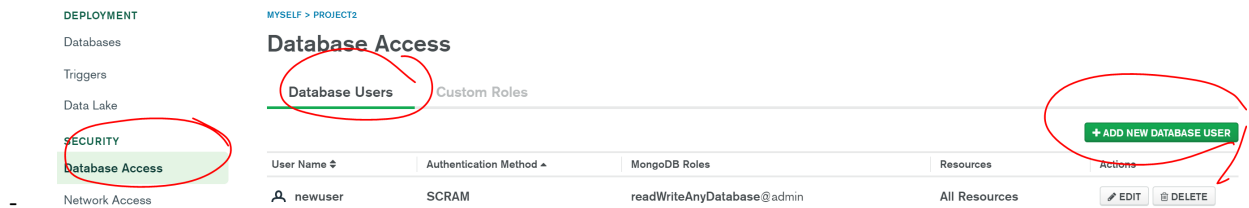


Once those steps have been completed, we can move on and get our connection information.



- Now add a user name and a password of your choice (e.g., rezadb, 12345)

- head to the **Database Access** and then the **MongoDB Users** and add a **New User** in order to connect to the **database**.



- You can select whatever **username** and **password** you want, but make sure you are an **Atlas Admin**. After which, you can just hit **Add**.

Add New Database User

Create a database user to grant an application or user, access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding [Access Manager](#).

Authentication Method

Password

Certificate

AWS IAM
(MongoDB 4.4 and up)

MongoDB uses **SCRAM** as its default authentication method.

Password Authentication

Database User Privileges

-
-
- At this point you will be transferred to Database Access Page
-

×

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're ready to connect. Choose how you want to connect in the next step.

1

Whitelist a connection IP address

✓ An IP address has been whitelisted. [Add another whitelist entry in the IP Whitelist tab.](#)

2

Create a Database 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.

Close

Choose a connection method

×

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 user security permission below.

1

Whitelist a connection IP address

✓ An IP address has been whitelisted. [Add another whitelist entry in the IP Whitelist tab.](#)

2

Create a Database User

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

Username

rezadb

Password

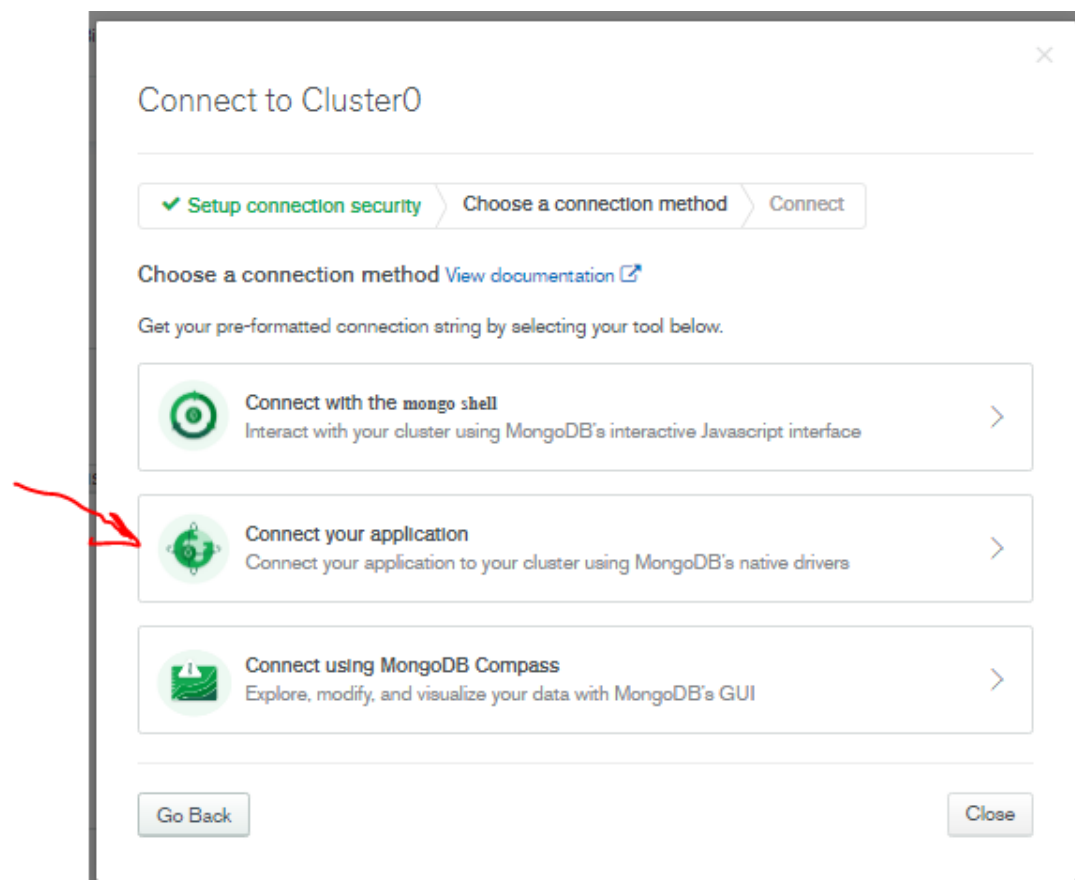
[Autogenerate Secure Password](#)

[SHOW](#)

Create Database User

Close

Choose a connection method



×

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

mongodb+srv://rezadb:<password>@cluster0.p1g9w.gcp.mongodb.net/<dbname>

Copy

<div>

Replace **<password>** with the password for the **rezadb** 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](#)

Go Back

Close