COMP2068 – JavaScript Frameworks

Lesson 5 – Intro to MongoDB and Mongoose

The Origins of NoSQL

- The NoSQL concept evolved as a response to the scaling problems encountered by web applications with relational databases
- In relational databases, data is stored throughout the database and reconstructed in the application logic through the creation of objects. This "Object-Relational Mapping" (ORM) requires significant design and coding work.
- The relational model is extremely inflexible…what happens when a new field needs to be added or the data rules change? What would an example be?

Relational data model

Highly-structured table organization with rigidly-defined data formats and record structure.

The NoSQL Alternative

Document data model

Collection of complex documents with arbitrary, nested data formats and varying "record" format.

- NoSQL databases offered developers another data storage mechanism: "documents" which usually store data in key-value pairs
- NoSQL documents are designed for high performance and scalability as they are schema-less: each "record" in the same document can actually have a different format and the format simply evolves over time without the requirement to modify the document structure first
- Documents can be nested inside other documents eliminating the need to create table relationships, foreign keys, and joins
- NoSQL's horizontal scaling makes it perfectly suited to cloud hosting solutions, which can automatically scaled based on user demand

A Sample MongoDB Document

- Each document represents a “record”
- Documents are stored in “collections”
- Different documents in the same collection can have different schemas (unlike records in a relational database table
- Documents are structured with key / value pairs
- A document field can contain another document, an array, or an array of other documents
- All documents must have a unique **_id** field as primary key. MongoDB automatically adds this field to every collection
- Further reading: https://docs.mongodb.org/getting-started/node/

```
{
  "_id" : ObjectId("54c955492b7c8eb21818bd09"),
   "address" : {
     "street" : "2 Avenue",
     "zipcode" : "10075",
     "building" : "1480",
     "coord" : [ -73.9557413, 40.7720266 ],
   },
   "borough" : "Manhattan",
   "cuisine" : "Italian",
   "grades" : [
     {
       "date" : ISODate("2014-10-01T00:00:00Z"),
       "grade" : "A",
       "score" : 11
     },
     {
       "date" : ISODate("2014-01-16T00:00:00Z"),
       "grade" : "B",
       "score" : 17
     }   ],
   "name" : "Vella",
   "restaurant_id" : "41704620“
}
```

What abut Relationships?

```
{
 "_id" : ObjectId("54c955492b7c8eb21818bd09"),
  "address" : {
     "street" : "2 Avenue",
     "zipcode" : "10075",
     "building" : "1480",
     "coord" : [ -73.9557413, 40.7720266 ],
  },
  "borough" : "Manhattan",
  "cuisine" : "Italian",
  "grades" : [
     {
        "date" : ISODate("2014-10-01T00:00:00Z"),
        "grade" : "A",
        "score" : 11
     },
     {
        "date" : ISODate("2014-01-16T00:00:00Z"),
        "grade" : "B",
        "score" : 17
     }   ],
  "name" : "Vella",
  "restaurant_id" : "41704620“
}
```

- NoSQL databases don’t use Foreign Keys or enforce relationship integrity like relational databases do
- Instead, they are some guidelines and options for modelling relationships between data
- A good summary of these guidelines can be found at https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1

Why MongoDB?

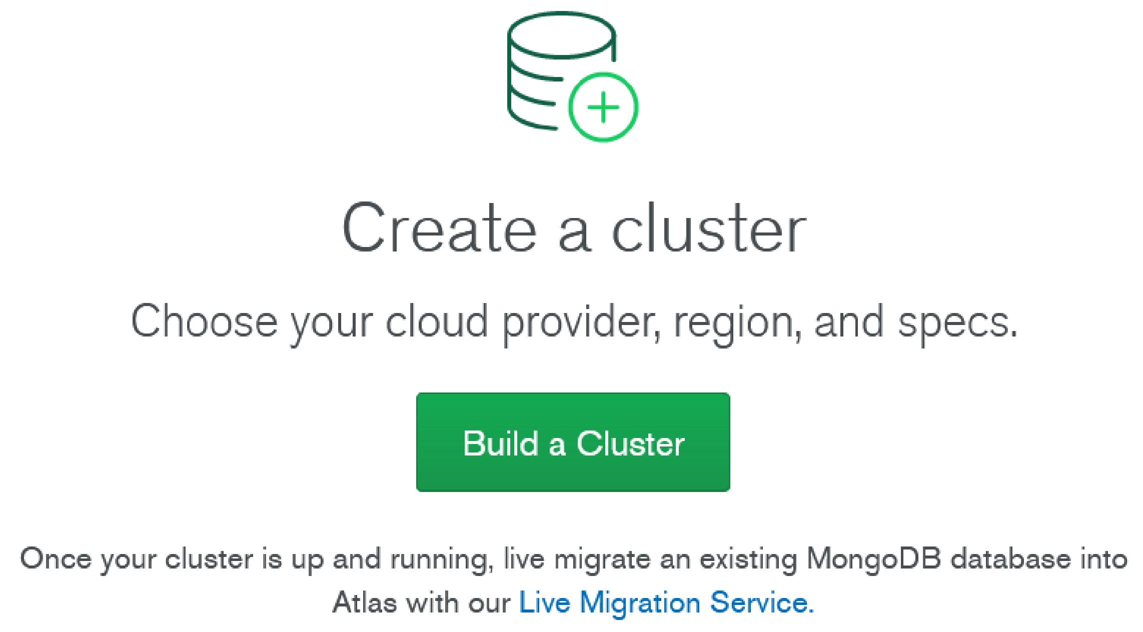
- Created in 2007 and coming from the word "humongous", MongoDB is the world's most popular NoSQL database
- Data storage in the form of BSON – Binary JSON. These key/value pairs are extremely fast and play well with Node and Express as both our application code and the data itself are in JavaScript format.
- Supported by most cloud hosting platforms
- Includes automatic unique ObjectID
- Supports adhoc queries with a very lightweight syntax:

```
db.posts.find({ title:/mongo/ });
```

How do I get MongoDB?

- Register at www.mongodb.com and download MongoDB's Compass GUI tool
- Register for MongoDB's Cloud Service "Atlas" to get 512 MB of database space for free
- You can also download and install MongoDB locally but you cannot use this for assignments as it's a local database

Setting up your cloud MongoDB

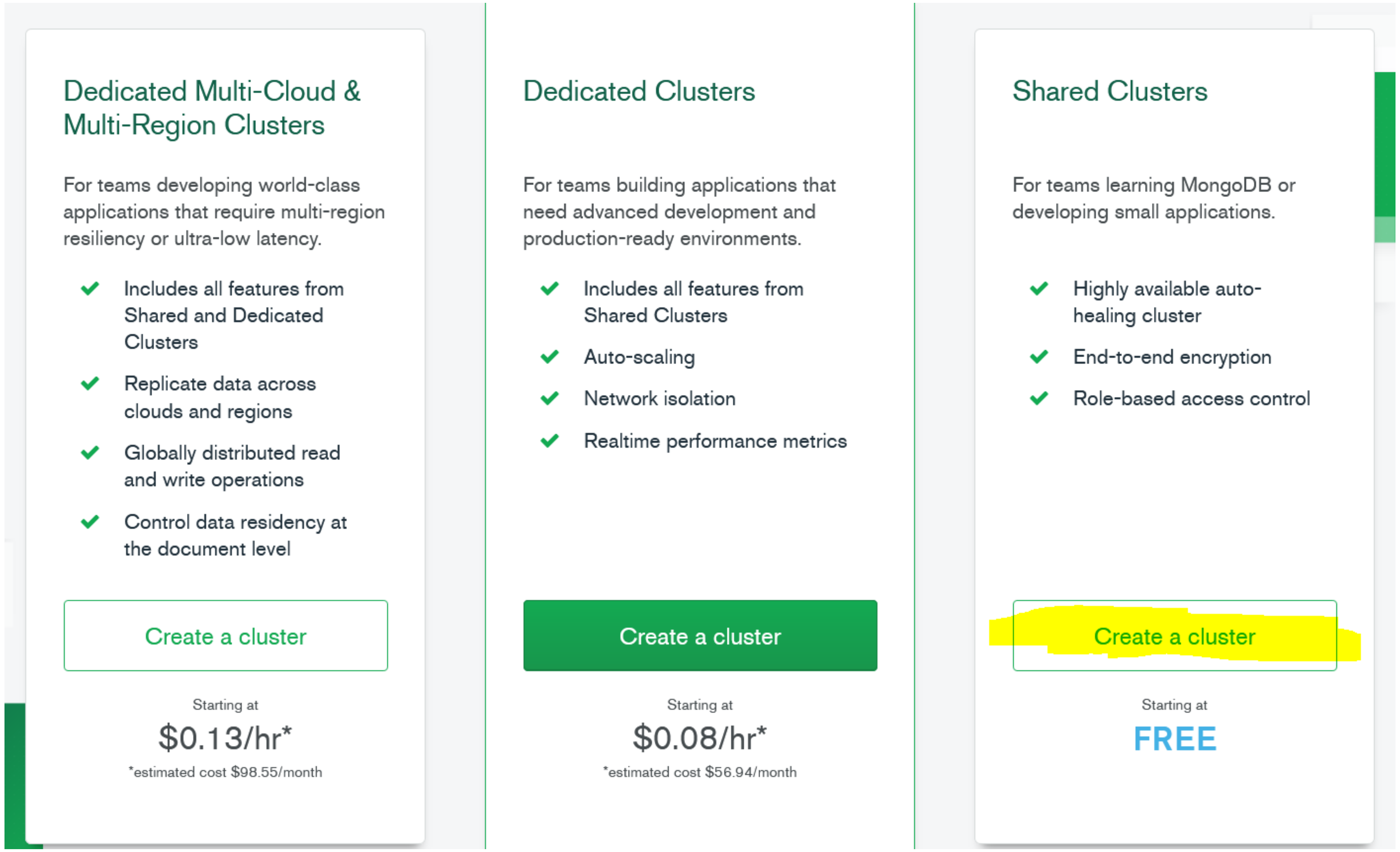

Create a cluster

Choose your cloud provider, region, and specs.

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

- Sign in at https://mongodb.com
- To create a free database (you can only create 1 free one per account), create a new **Cluster**
- You can use this same Cluster for both our in-class app and your Assignment 2

Select Free Shared Cluster



Dedicated Multi-Cloud & 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 clouds and regions
- ✓ Globally distributed read and write operations
- ✓ Control data residency at the document level

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

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

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

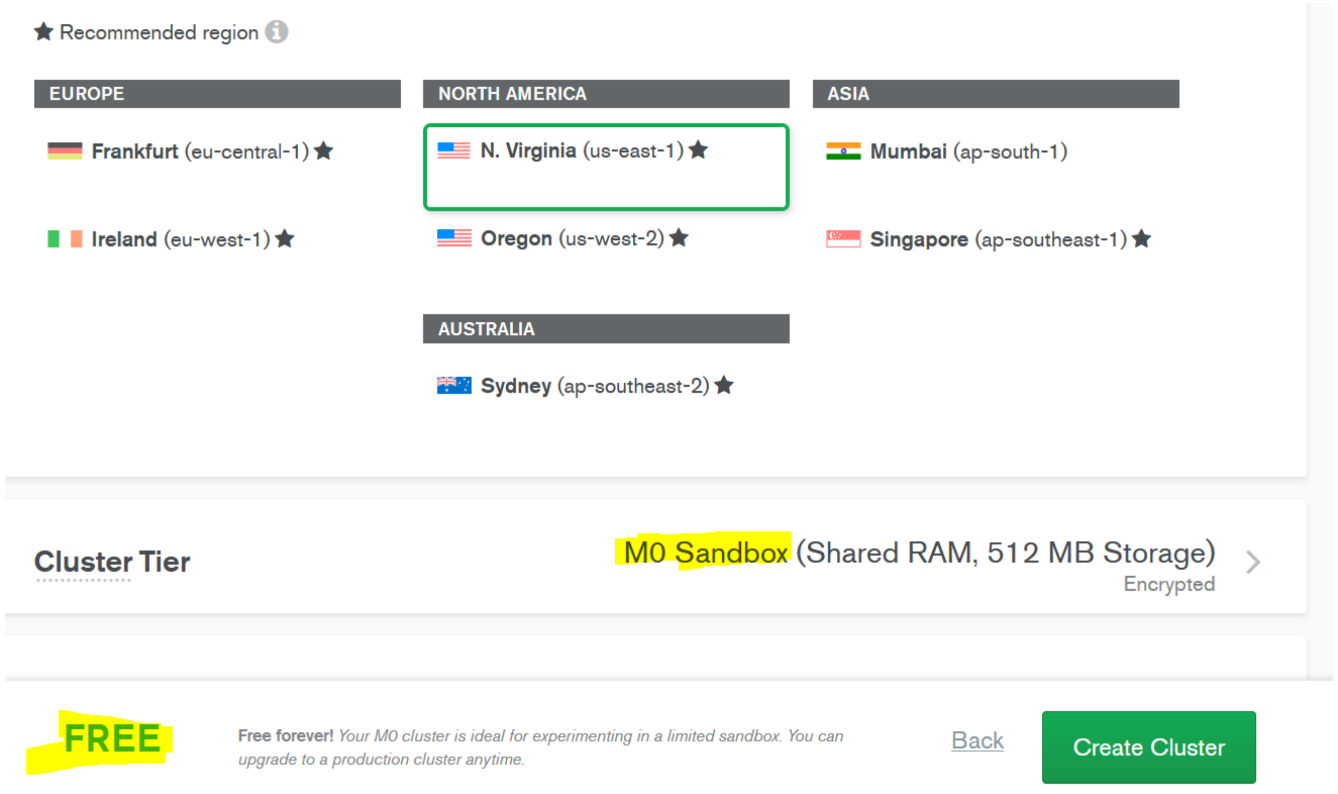
Shared Clusters

For teams learning MongoDB or developing small applications.

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

Starting at
FREE

Using the Free Sandbox



★ Recommended region

EUROPE	NORTH AMERICA	ASIA
Frankfurt (eu-central-1) ★	**N. Virginia** (us-east-1) ★	**Mumbai** (ap-south-1)
Ireland (eu-west-1) ★	**Oregon** (us-west-2) ★	**Singapore** (ap-southeast-1) ★

AUSTRALIA
Sydney (ap-southeast-2) ★

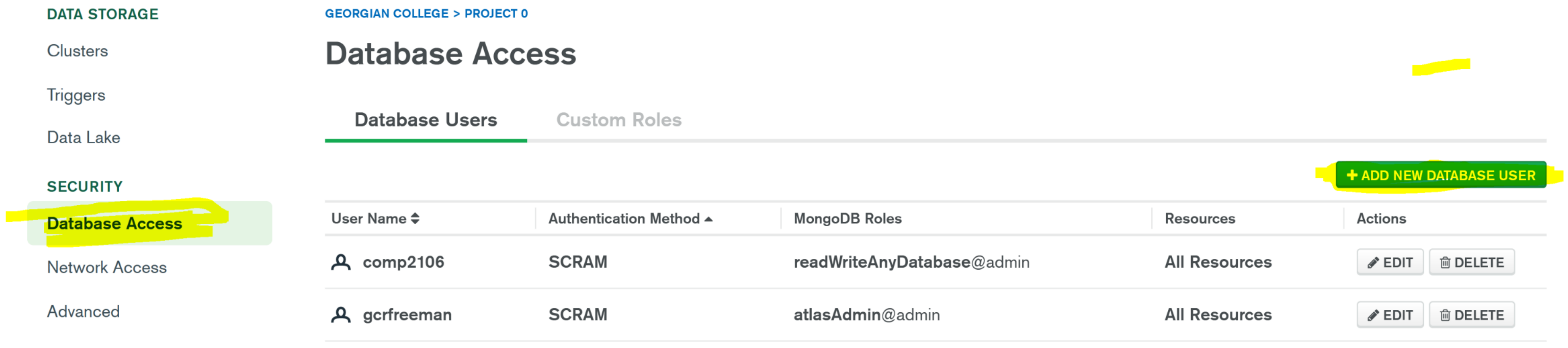
Cluster Tier M0 Sandbox (Shared RAM, 512 MB Storage)
Encrypted

FREE

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

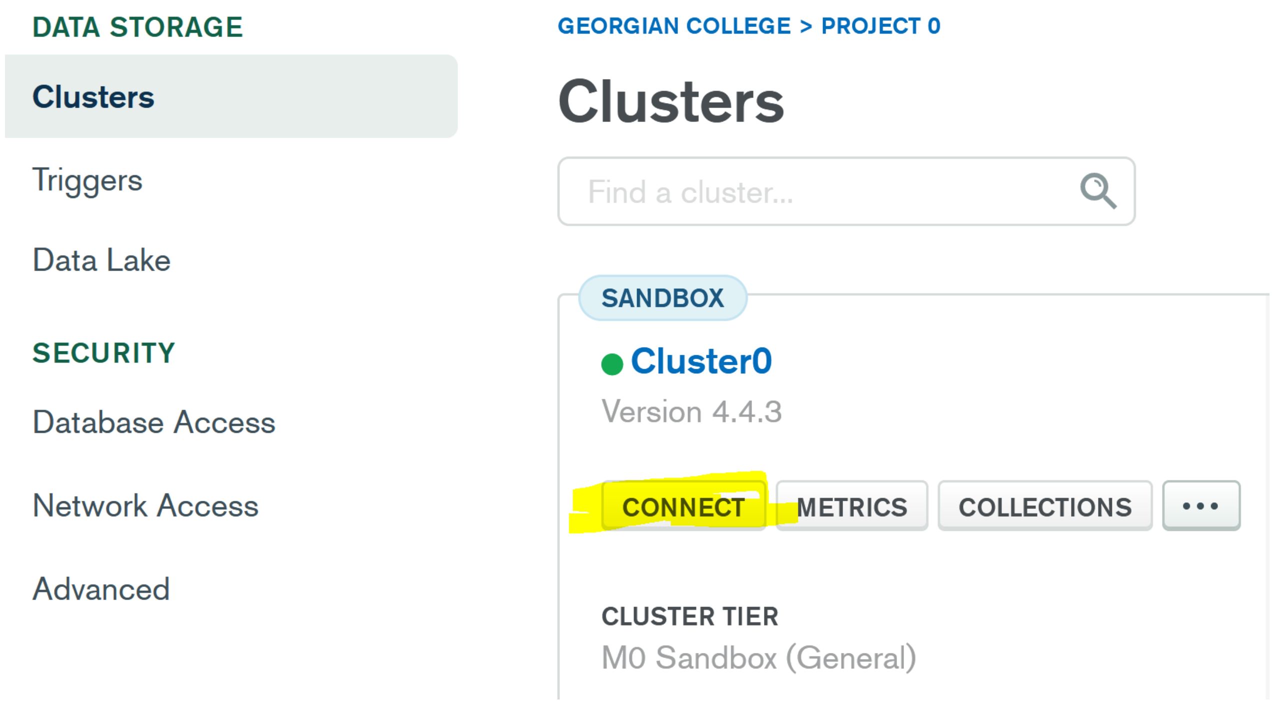
- Accept the default options
- Ensure your tier says "Sandbox" and "Free"

Create a User Account



User Name	Authentication Method	MongoDB Roles	Resources	Actions
comp2106	SCRAM	readWriteAnyDatabase@admin	All Resources	EDIT DELETE
gcrfreeman	SCRAM	atlasAdmin@admin	All Resources	EDIT DELETE

Connect to your Cluster



DATA STORAGE

Clusters

Triggers

Data Lake

SECURITY

Database Access

Network Access

Advanced

GEORGIAN COLLEGE > PROJECT 0

Clusters

Find a cluster...

SANDBOX

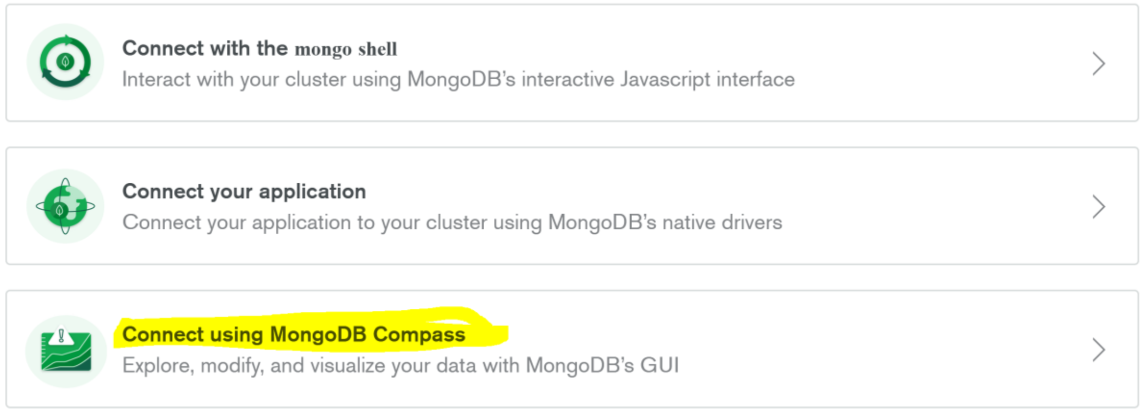
● Cluster0

Version 4.4.3

CONNECT METRICS COLLECTIONS •••

CLUSTER TIER

M0 Sandbox (General)



Connect with the mongo shell
Interact with your cluster using MongoDB's interactive Javascript interface

Connect your application
Connect your application to your cluster using MongoDB's native drivers

Connect using MongoDB Compass
Explore, modify, and visualize your data with MongoDB's GUI

CLI CRUD Operations with MongoDB

- First we need to connect to a cluster (or create a new one)
- Mongo comes with a “test” database pre-installed locally, so to select it:

```
use test;
```

- To create documents:

```
db.beers.insert( { name: “Steamwhistle”, type: “Pilsner”} );
db.beers.insert( { name: “Sleeman Cream Ale”, type: “Ale”, onSale: true } );
```

- Notice we neither had to create the beers collection first, nor define any schema for it. In fact, our 2 documents have slightly different schemas

CRUD Operations with MongoDB

- Query the data:

```
db.beers.find().pretty();
```

- Notice the unique _id field of type ObjectId that Mongo auto-generates
- Update a document
- By default Update modifies a single document, though there is a multi option:

```
db.beers.update( {name: “Sleeman Cream Ale”,
                 $set: { onSale: false } } );
```

- Adding `multi: true` to the above would update multiple records

CRUD Operations with MongoDB

- To delete documents:

```
db.beers.remove( { name: “Steamwhistle” } );
```

- The remove() function takes an optional parameter which specifies how many documents should be deleted

Introducing Mongoose

How does Node talk to MongoDB? The most common way is via the Mongoose npm package

Find out more at http://www.mongoosejs.com
(Click the arrow when in Slide Show mode)

Mongoose

- Mongoose is an Object – Document Mapper (ODM)

- In the words of the Mongoose team:

  “Mongoose provides a straight-forward, schema-based solution to model your application data. It includes built-in **type casting**, **validation**, **query building**, **business logic hooks** and more, out of the box.”

- Mongoose helps our Node apps enable MVC patterning

```
const mongoose = require('mongoose');
mongoose.connect('mongodb+srv://<username>:<password>@host/database');

const Cat = mongoose.model('Cat', { name: String });

var kitty = new Cat({ name: 'Zildjian' });
kitty.save(function (err) {
  if (err) // ...
  console.log('meow');
});
```

mongoose

elegant mongodb object modeling for node.js

Step 1: Add some test data

- To add article documents:

```
db.articles.insert( { created: “10/1/2015”,
                      title: “Our First Article”,
                      content: “This will be really interesting some day.” } );

db.articles.insert( { created: “10/2/2015”,
                      title: “Here is number two!”,
                      content: “This is even more fascinating.” } );

db.articles.insert( { created: “10/3/2015”,
                      title: “Day 3: Article 3”,
                      content: “OK, this one is rather boring.” } );
```

Step 2: Connect in app.js (for now)

- In app.js, we need to link to the mongoose package, then try connecting:

```
const mongoose = require(“mongoose”)
mongoose.connect(“mongodb+srv://<username>:<password>@host/database”, {

        useNewUrlParser: true,

        useUnifiedTopology: true

}).then({

        (res) => { console.log(‘Connected’)

}).catch(() => {

        console.log(‘Connection Error’)

})
```

- Let nodemon restart your app then check the console for a message

Step 3: Build a mongoose schema

- We need to build a schema or model to represent the games collection within our Node application – **models/game.js** (notice the name is singular by convention)
- Our model needs to require mongoose
- We instantiate a new schema
- Then we define the schema in JSON format – what does a document in this collection look like?
- Make the model visible to other parts of the application

Step 4: Write the Read method in the Route

- Mongoose models have built-in CRUD methods we can leverage rather than writing our own
- Let's build an articles page that uses the schema defined in **models/game.js** to retrieve and display a list of games
- Create **controllers/games.js** and add a GET handler for the path **/games**
- Link to this new route in **app.js**
- Create **views/games/index.hbs** to display the data.

Step 5: Adding new data

- Let’s build an input form at **views/games/add.hbs**
- Then we need a GET handler inside **controllers/games.js** to load the form. Let’s also add a link to this page at the top of our **views/games/index.hbs**
- Now we need a POST handler in **controllers/games.js** to save the new record and redirect back to the main games page