



Aplicación con acceso a BD

Los datos en MongoDB se almacenan en un clúster. Un **clúster** es una agrupación de ordenadores, a menudo llamados en este contexto nodos. Los datos de cada colección se reparten entre los nodos, logrando así el soporte de cantidades masivas de datos.

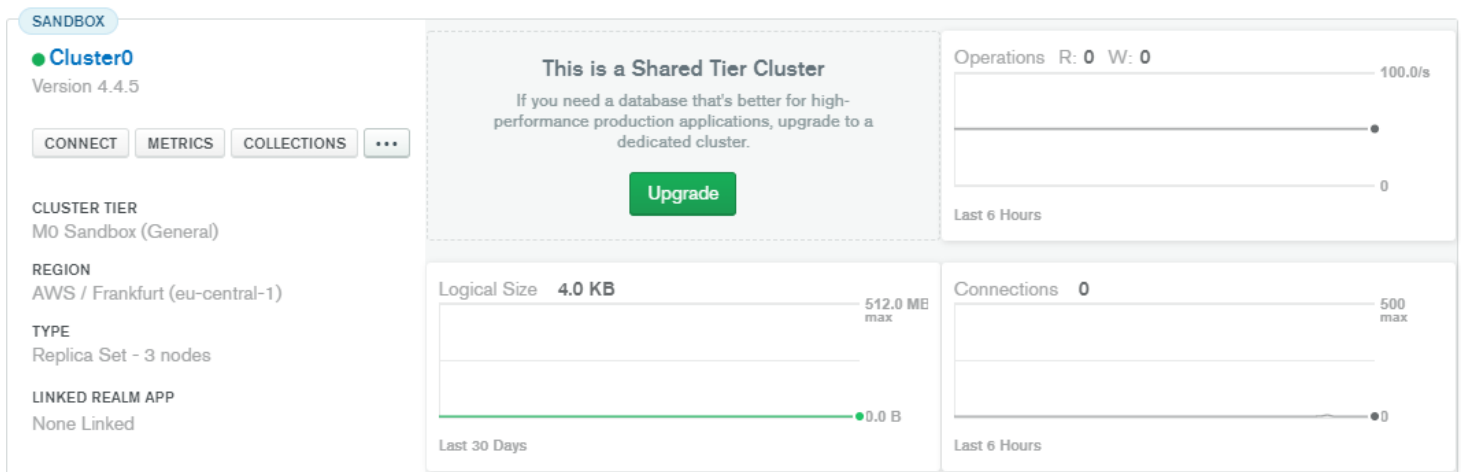
Para crear un cluster, accedemos a la página MongoAtlas

https://www.mongodb.com/cloud/atlas/lp/try2?utm_source=google&utm_campaign=gs_emea_spain_search_core_brand_atlas_desktop&utm_term=mongo%20atlas&utm_medium=cpc_paid_search&utm_ad=e&utm_ad_campaign_id=12212624563&gclid=CjwKCAjwjuqDBhAGEiwAdX2cj3P6Qt1NOc2cjdz025qLKnwga5MJTCvwMu9w7yos8ovETReEXIORtxoCw7gQAvD_BwE

Clicquemos en la opción acceder con google   creamos nuestra cuenta accediendo con nuestro correo y contraseña en el enlace Sing up. Si es la primera vez, deberemos verificar la cuenta con el mensaje que nos enviará a nuestro correo. Ahora, seguiremos una serie de pasos:

- 1º- Clicquemos en create un cluster.
- 2º- Creamos la versión gratuita.
- 3º- Elegimos Amazon como proveedor
- 4º- La región de Virginia
- 5º- El tipo de cluster será Sandbox, un entorno de pruebas
- 6º- Lo nombramos y le damos a crear.

Ya tendríamos creado el cluster:



Restricciones de acceso

Accedemos a Network access. Aquí nos muestra las direcciones ip que pueden acceder al Cluster. Agregaremos la de nuestro domicilio personal.

SECURITY

Database Access

Network Access

Advanced



+ ADD IP ADDRESS

Accedemos a la línea de comandos de Windows escribiendo cmd en la ventana ejecutable. Acto seguidos se nos abrirá el CLI y pondremos el comando `ipconfig`. Nos mostrará los datos de nuestra ip; copiamos nuestra dirección. Luego cliqueamos en el botón add ip address e introducimos los datos.

Si quisiéramos que se pudiese acceder desde cualquier ip, tenemos la opción allow access from anywhere, aunque no es recomendable.

IP Address	Comment	Status	Actions
192.168.1.106/32	MoronCasa	● Active	<button>⚙ EDIT</button> <button>🗑 DELETE</button>
185.197.88.21/32	PuntaDelVerde	● Active	<button>⚙ EDIT</button> <button>🗑 DELETE</button>

Creación de usuarios

Procedemos a crear nuestro usuario, otorgándonos todos los permisos y permitiéndonos la manipulación de los datos. Accedemos a Database Acces, y cliqueamos en add new database user.

SECURITY

Database Access

Network Access

Advanced

Hay tres opciones para verificar al usuario, elegiremos la más clásica; mediante una password. En las opciones de Authentication, escribimos el nombre del usuario y su clave, seguido de sus privilegios, que serán de lectura y escritura. Finaliamos con el add user.

Database Access

Database Users

Custom Roles

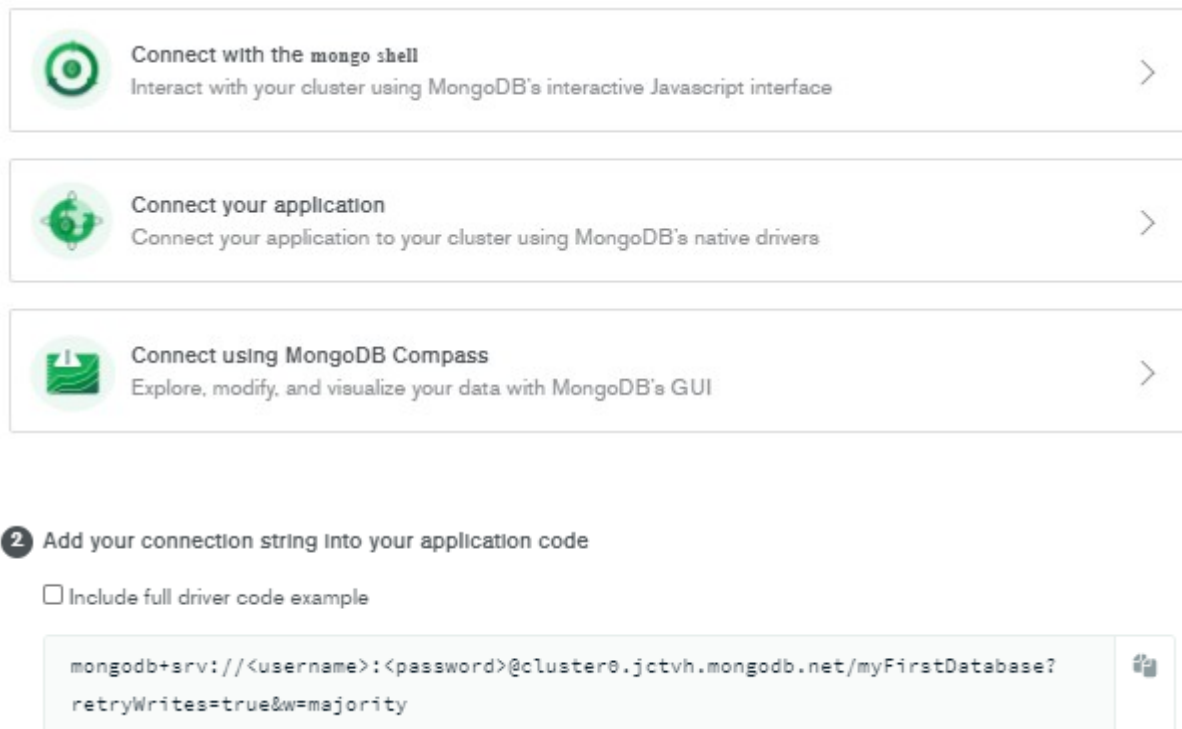
+ ADD NEW DATABASE USER				
User Name ↕	Authentication Method ▲	MongoDB Roles	Resources	Actions
👤 marmol	SCRAM	readWriteAnyDatabase@admin	All Resources	<button>✎ EDIT</button> <button>🗑 DELETE</button>

Es recomendable tener nuestra cuenta personal con todos los privilegios, y otra invitada con permisos de lectura, que se usará para exponer los trabajos.

Conexión con la base de datos

Accedemos a replit y creamos el proyecto tal y como se explicó en el primer ejercicio, que servía como introducción.

Crearemos un nuevo directorio y un archivo ts llamado database. Creamos la clase DataBase, cuyo atributo cadenaConexion será un String con el enlace de conexión a la base de datos del atlas. Para obtener el enlace, seleccionamos la opción connect del Clouster, luego en connect your application y luego copiamos el mensaje, sustituyendo el usuario, la clave y el nombre de la base de datos



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

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://<username>:<password>@cluster0.jctvh.mongodb.net/myFirstDatabase?
retryWrites=true&w=majority
```

Las otras dos opciones son para usar mongo shell y mongo compass. La práctica es similar al lo recién explicado: seleccionas el enlace y lo pegas en el mongo shell o en la conexión del compass, dependiendo de qué opción se ha elegido, y se sustituyen los datos.

Mi base de datos se llama MarmolCenter, y es el usado en el proyecto del 2º trimestre: <https://github.com/MarmolMunozAlejandro/Proyecto-final-2-trimestre.git>.

Después del atributo cadenaConexion, se crean dos métodos para conectar y desconectarse de la base de datos. Son métodos async que usa await:

- **async**: indica es que este método se quiere sincronizar con métodos que se ejecutarán de forma asíncrona. Sincronizarse con un método asíncrono significa que esperará a que termine.
- **await**: permite que un método que ha llamado a otro asíncrono se espere a que dicho método asíncrono termine. Complementa al async.

```

conectarBD = async () => {
  const promise = new Promise<string>( async (resolve, reject) => {
    await mongoose.connect(this._cadenaConexion, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
      useCreateIndex: true, // Para que cree el índice único asociado al campo unique
      useFindAndModify: false // impedimos modificar la base de datos
    })
    .then( () => resolve(`Conectado a ${this._cadenaConexion}`) )
    .catch( (error) => reject(`Error conectando a ${this._cadenaConexion}: ${error}`) )
  })
  return promise
}

```

```

desconectarBD = async () => {
  const promise = new Promise<string>( async (resolve, reject) => {
    await mongoose.disconnect()
    .then( () => resolve(`Desconectado de ${this._cadenaConexion}`) )
    .catch( (error) => reject(`Error desconectando de ${this._cadenaConexion}: $
    {error}`) )
  })
  return promise
}

```

Luego exportamos la base de datos.

```
export const db = new DataBase()
```

Ya podemos usar la base de datos. Ahora procedemos a crear los Schemas. Un esquema en Mongoose es una estructura JSON que contiene información acerca de las propiedades de un documento. Puede también contener información acerca de la validación y de los valores por default, y si una propiedad en particular es requerida. Los esquemas pueden contener lógica y otro tipo de información importante. Se crearán en una carpeta llamada model.

Para este ejercicio solo se usará la colección inversores.

```

import {Schema, model } from 'mongoose'

const inversorSchema = new Schema({
  id: Number,
  nombre: String,
  abono: Number
})

export const Inversores = model('inversores',inversorSchema)

```

Se ha creado la estructura json con los campos de la coleccion y se ha exportado

Ahora, en el index importamos el Schema, la conexión a la base de datos, los parámetros request y response, el método express y el puerto, que en este caso será 3000.

Reques un objeto que contiene información sobre la solicitud HTTP que provocó el evento. Se usa res para devolver la respuesta HTTP deseada.

```
import { Inversores } from './model/inversores'
import { db } from './database/database'
import { Request, Response } from 'express'
import express from 'express'
const app = express()
const port = 3000
```

Ahora podemos crear las funciones que deseemos, y llamarlas con el app.get(). En el ejercicio hay cuatro ejemplos, tres find y un aggregate.