TD2

Database with MongoDB

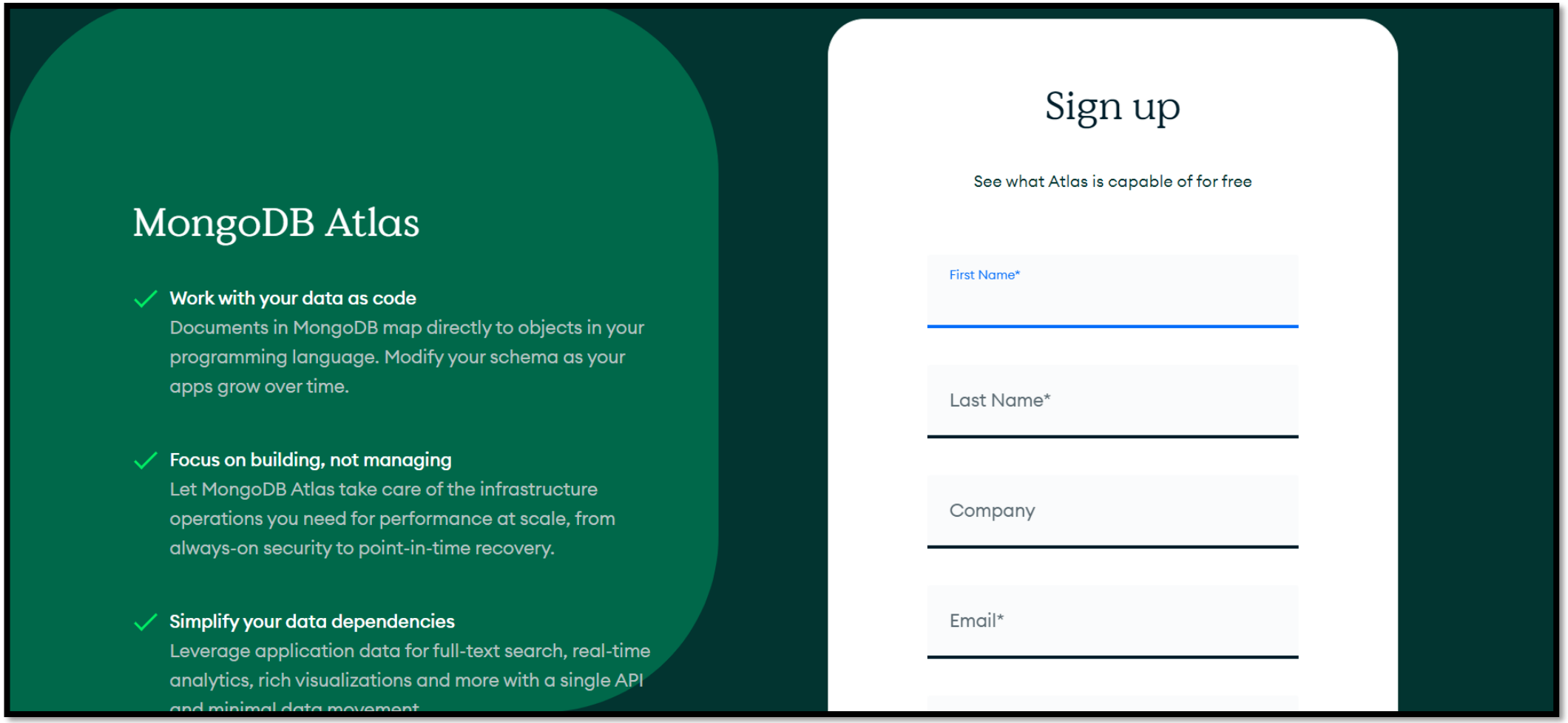
mongoose

https://www.jordansablon.fr/enseignement

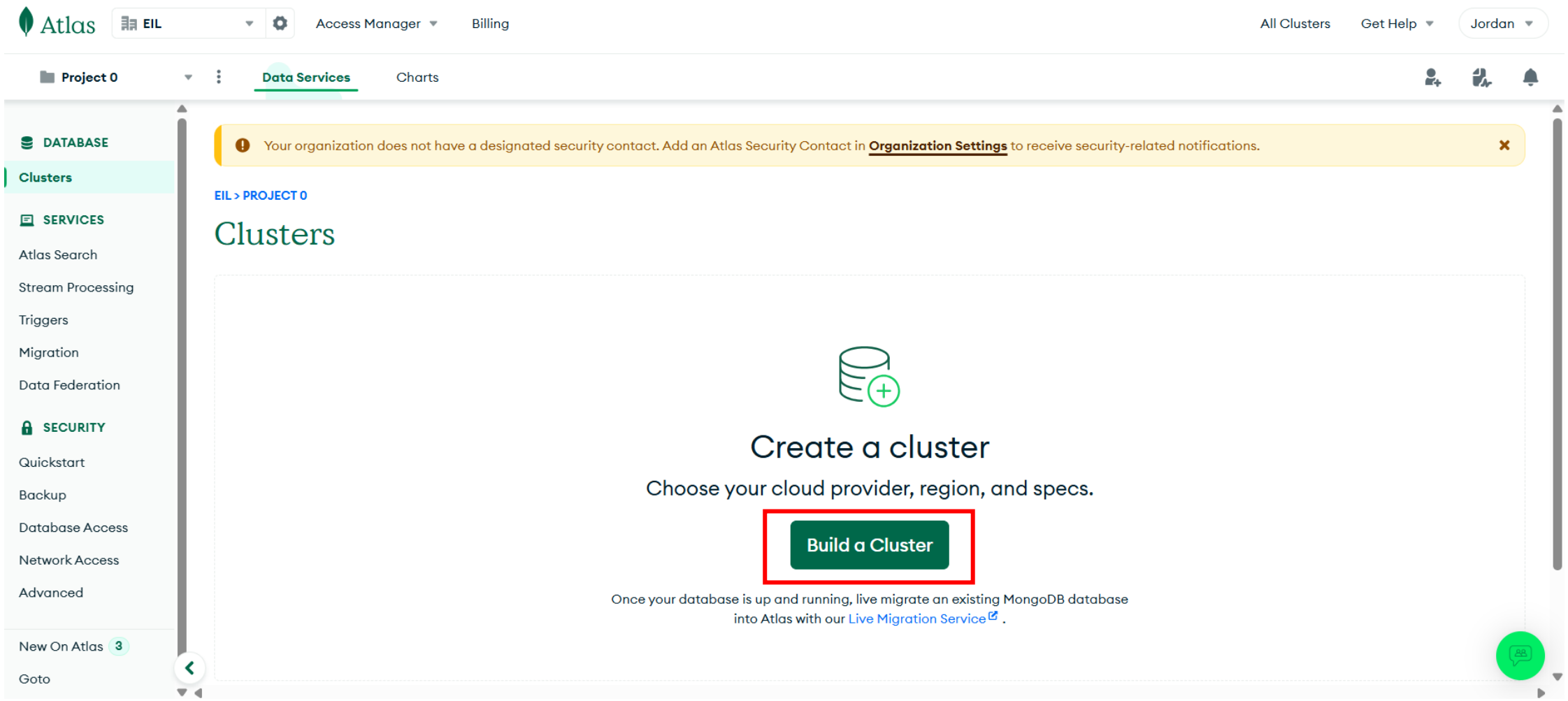
https://github.com/jsablonEnseignement/td2_sources

Sommaire

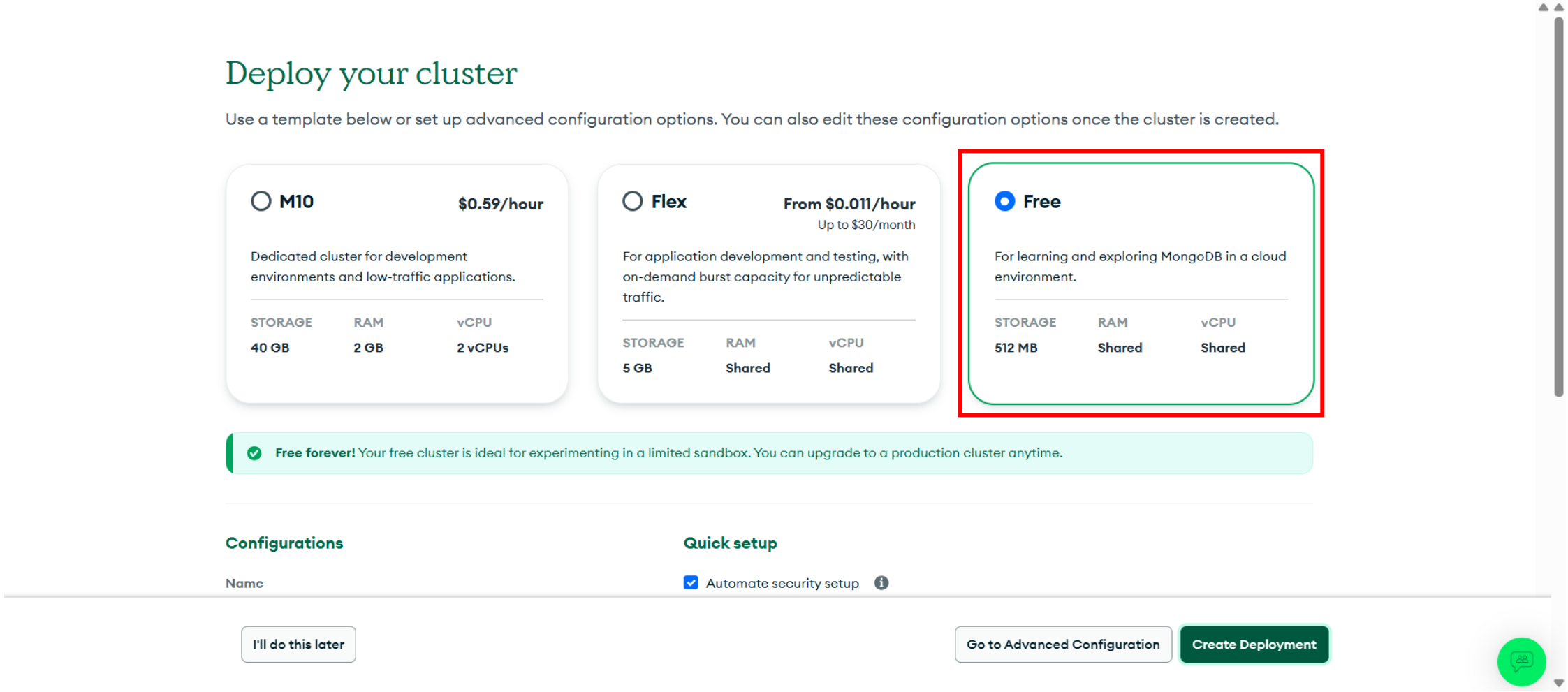
Créez un compte sur [Atlas]



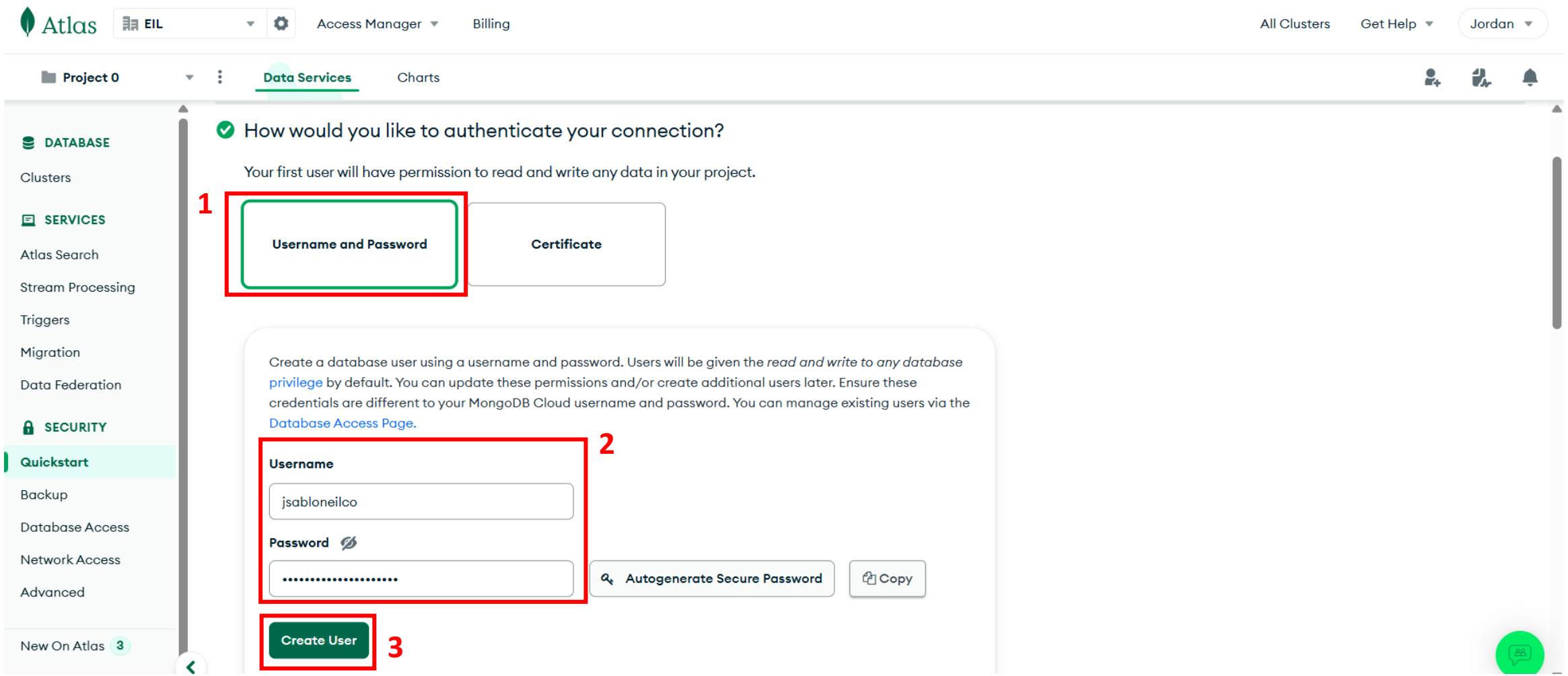
Sélectionnez « Build a Database » pour créer votre cluster de base de données



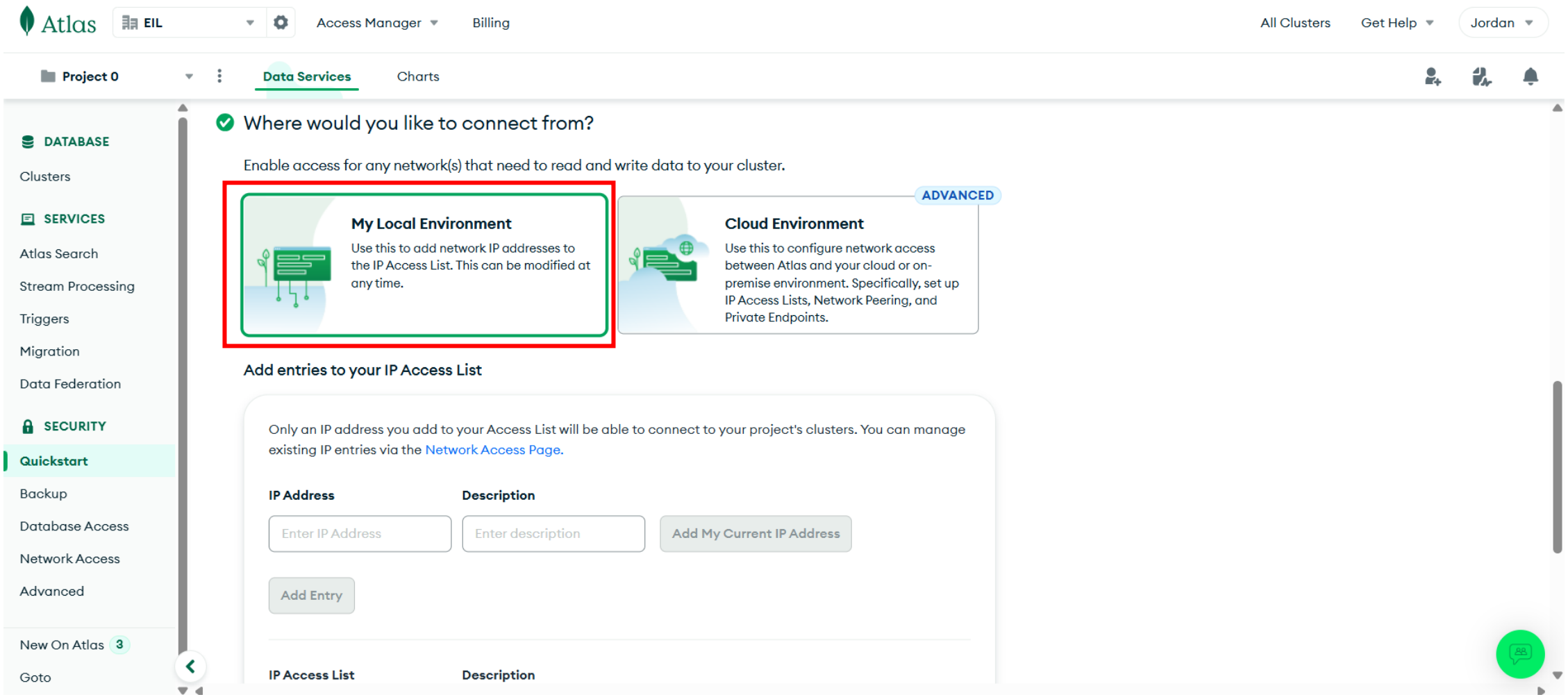
Sélectionnez le cluster « M0 » qui est gratuit



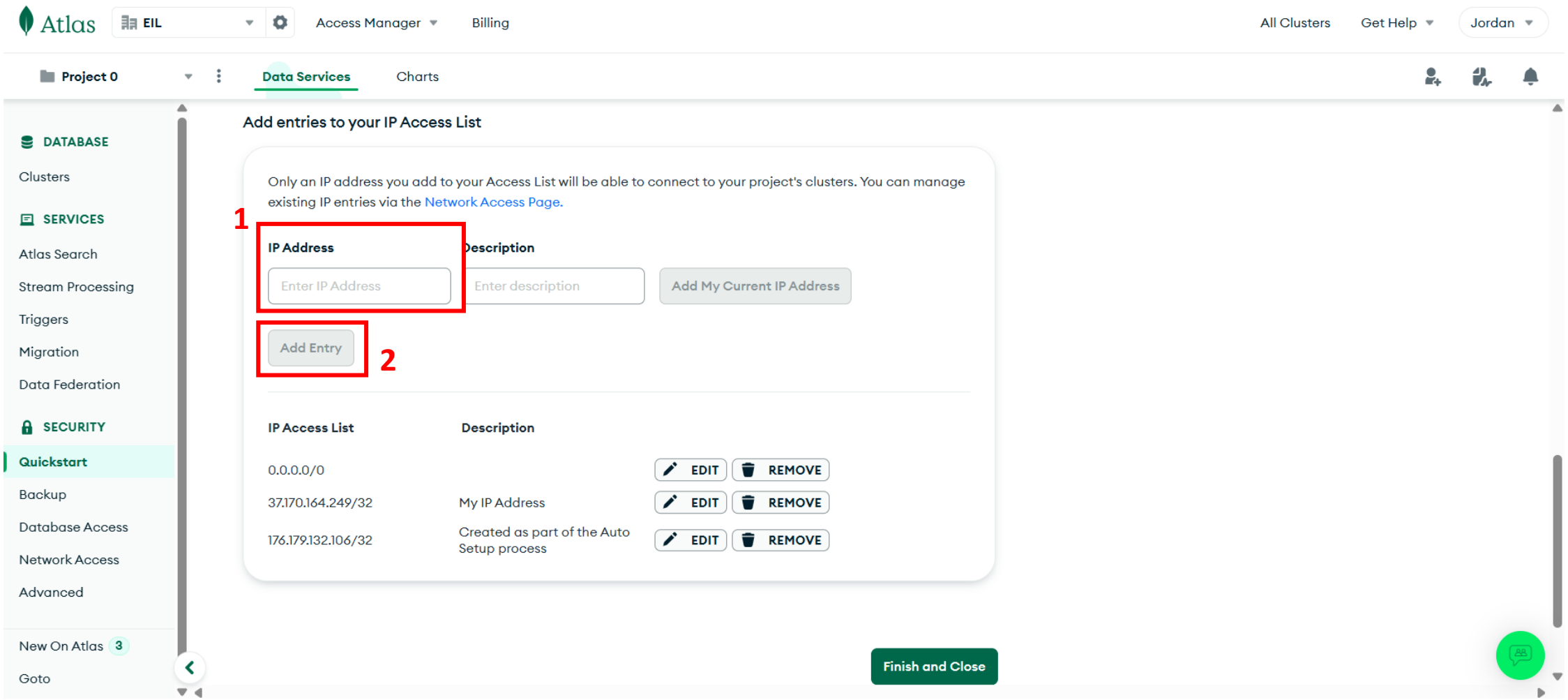
Descendre sur la page et créer un utilisateur pour vous connecter à votre base de données



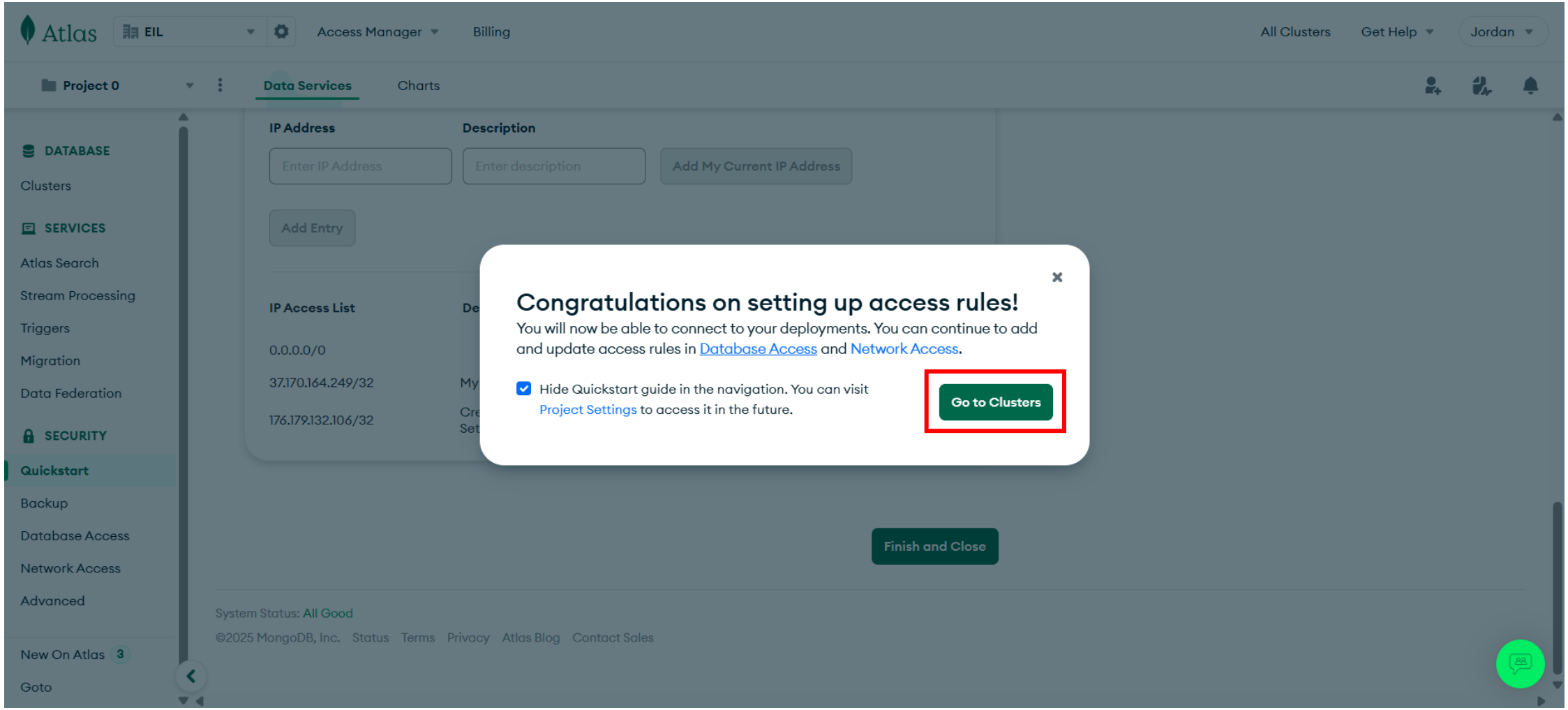
Descendre sur la page et sélectionnez ensuite « My Local Environment »



Descendre sur la page et entrez l'adresse IP 0.0.0.0/0 et cliquez sur « Add Entry »

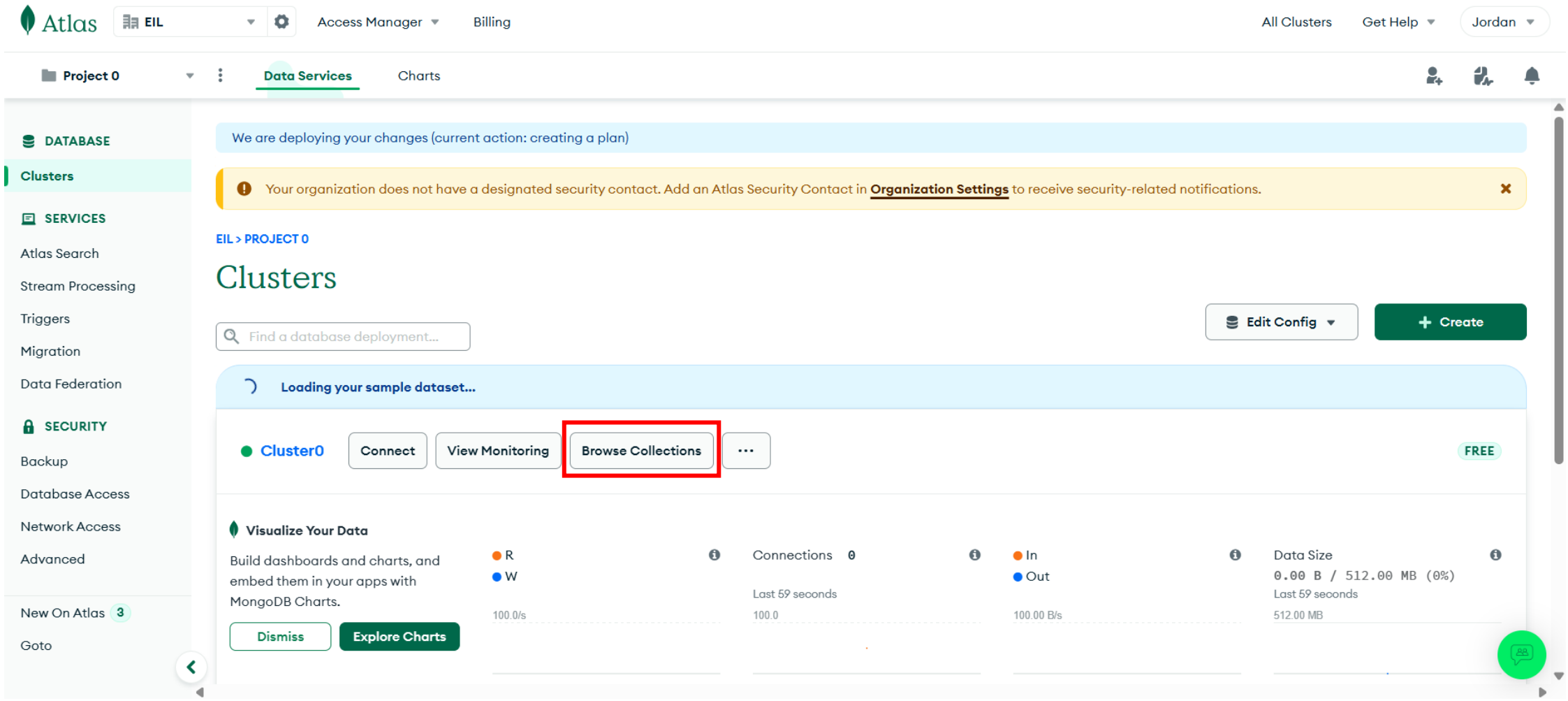


Cliquez sur « Finish and Close » : votre cluster de base de données est désormais créé

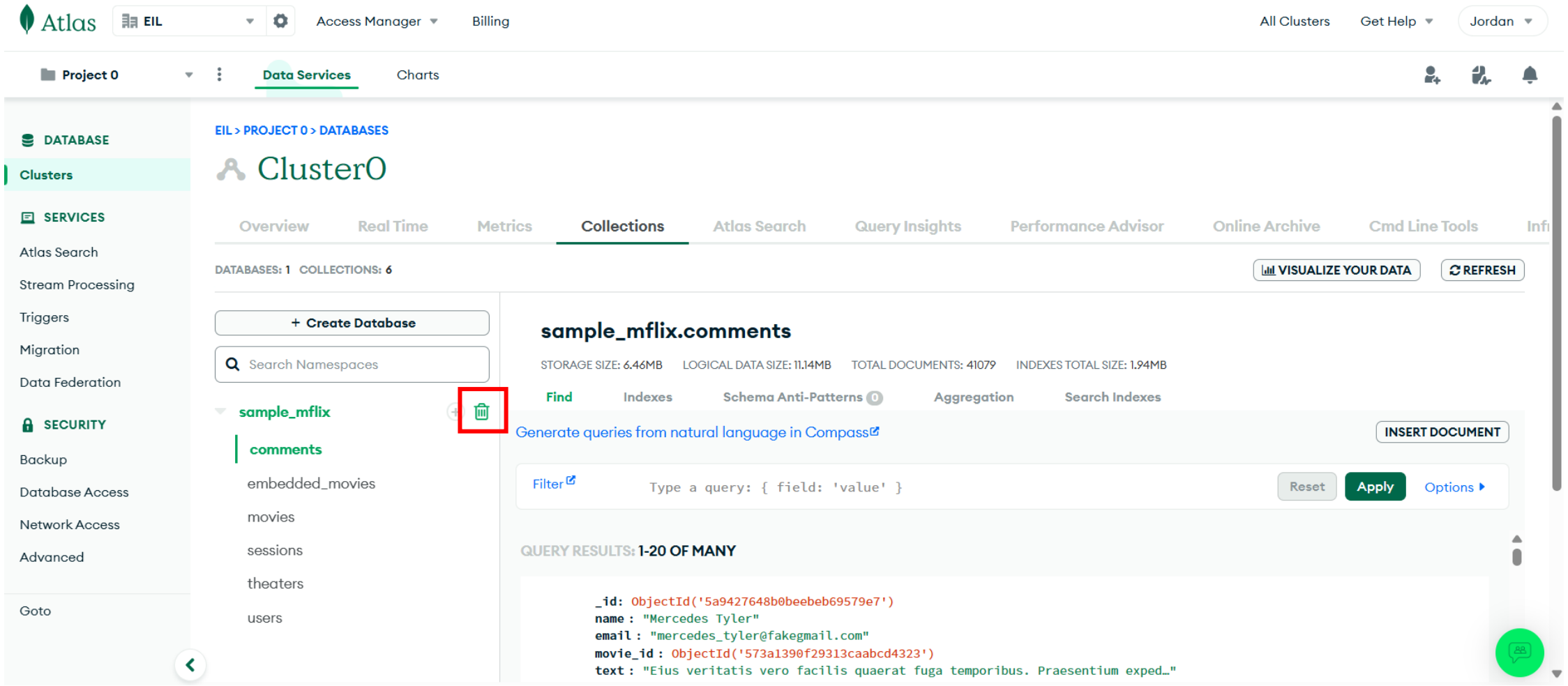


Initialisation de la collection

Depuis le menu « Clusters », cliquez sur « Browse Collections » pour afficher vos collections

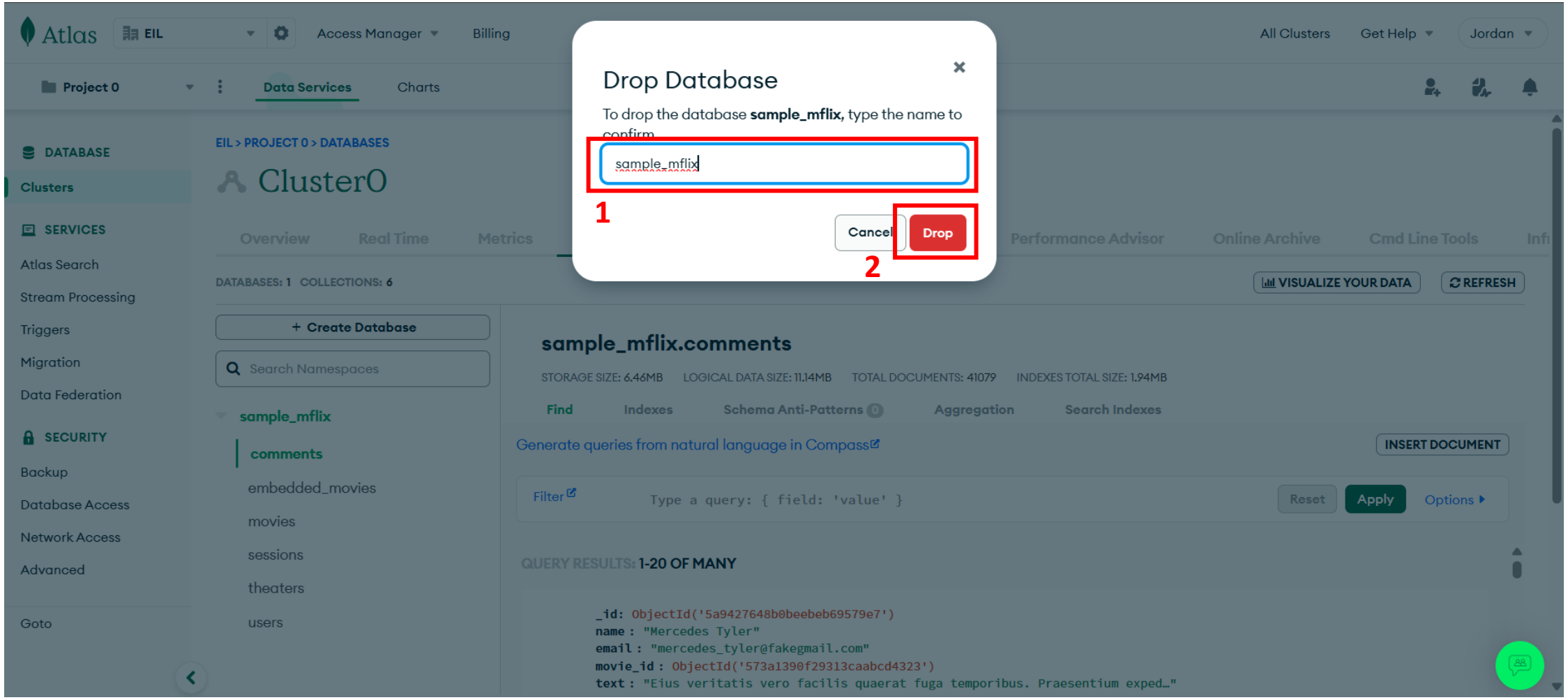


Supprimez la base de données d'exemple

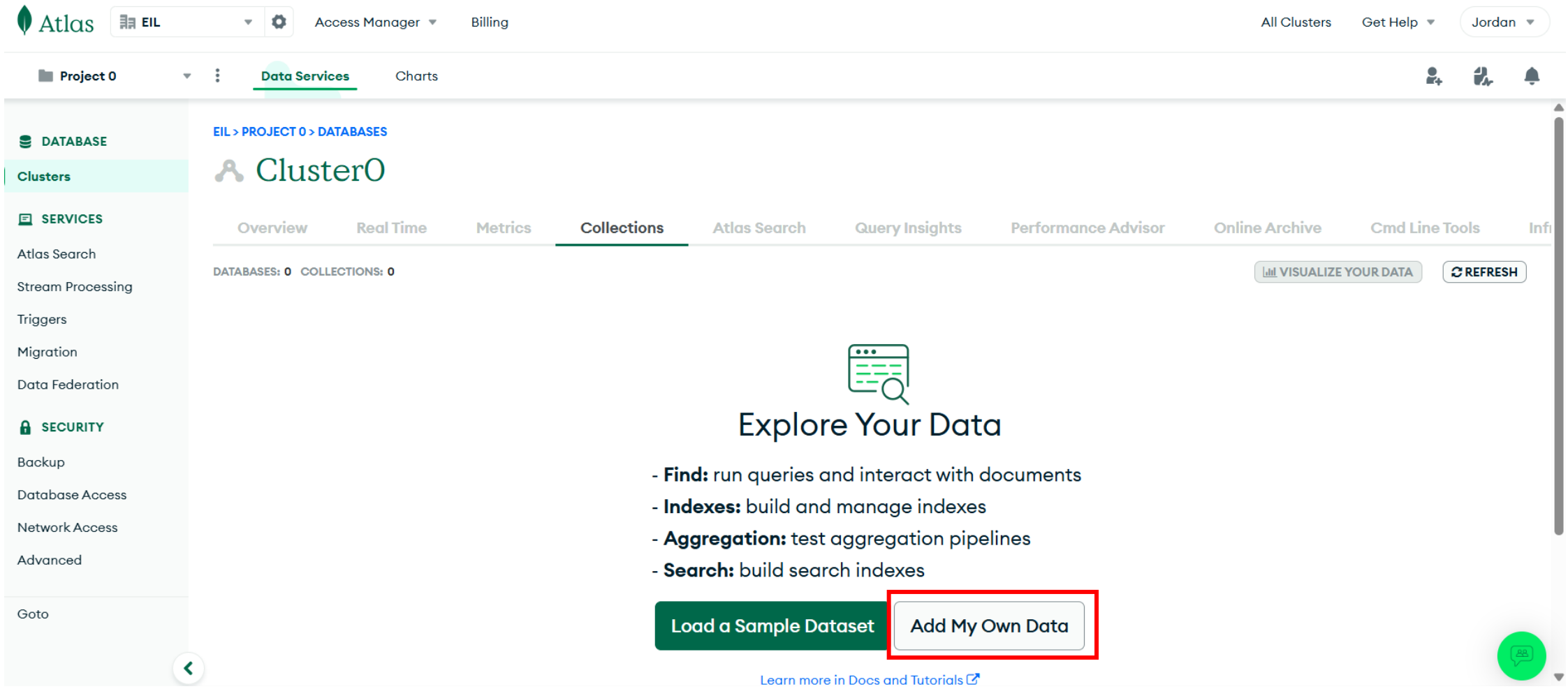


Initialisation de la collection

Validez la suppression en saisissant le nom de la base de données puis cliquez sur « Drop »

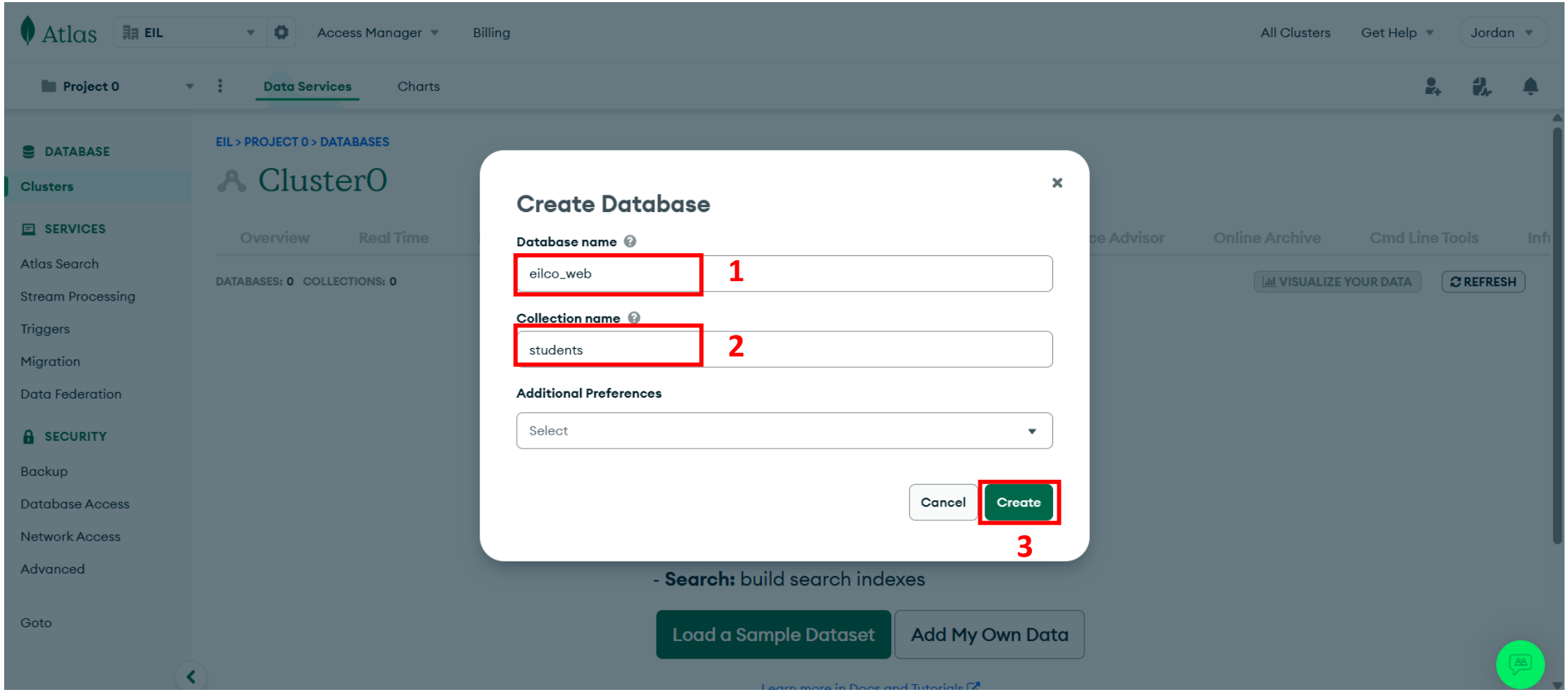


Cliquez sur « Add My Own Data »



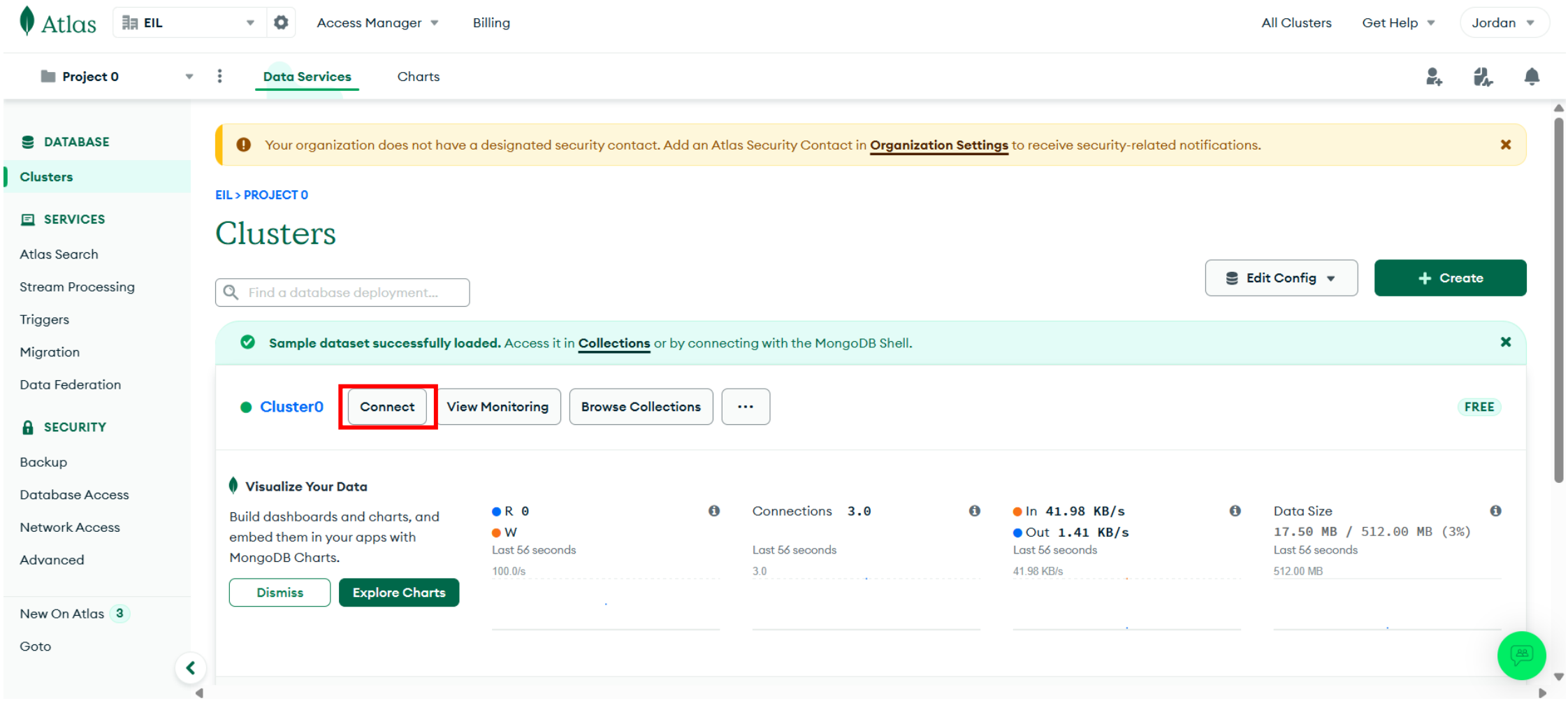
Initialisation de la collection

Renseignez les champs puis cliquez sur « Create » : votre collection est créée

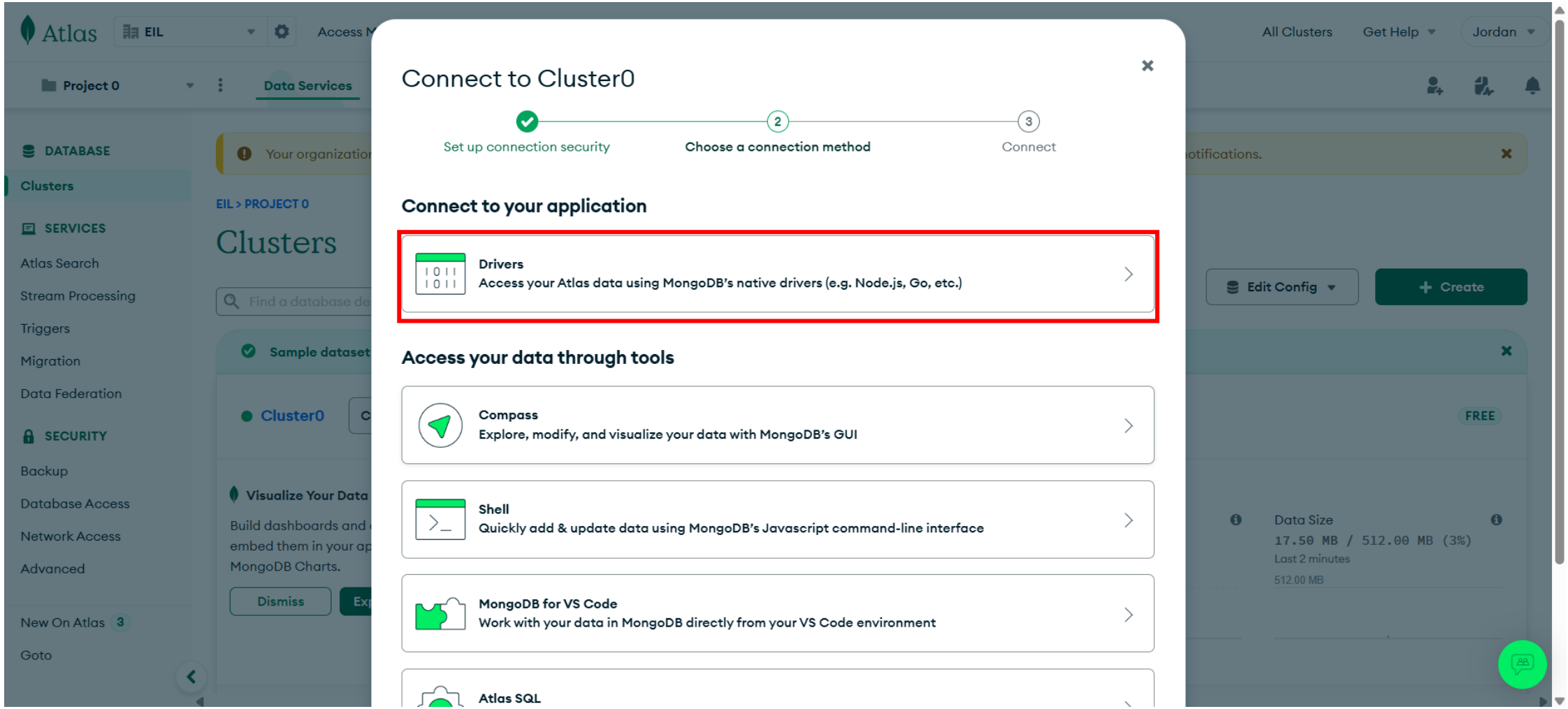


Insertion des données via MongoDB Compass

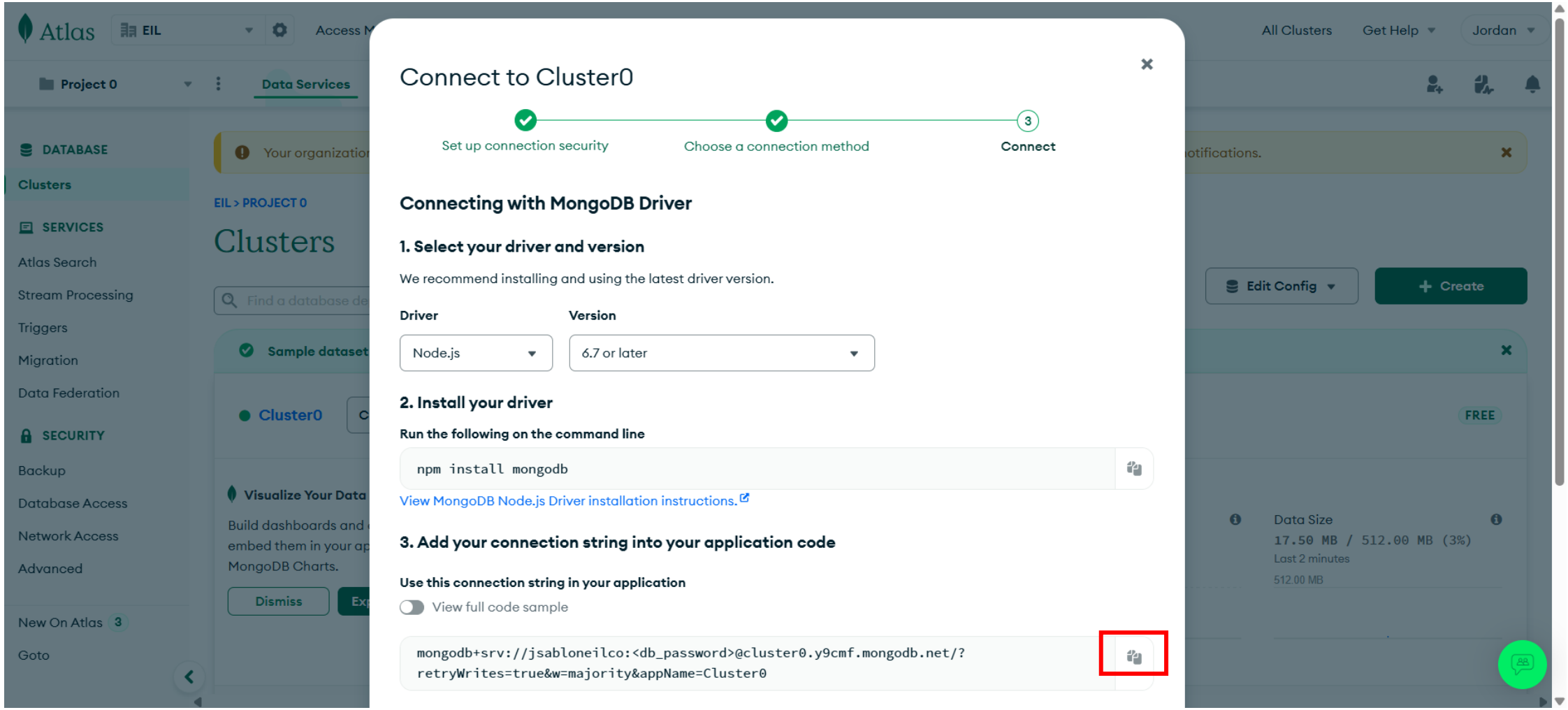
Depuis le menu « Clusters », cliquez sur « Connect » pour récupérer l'URL de connexion



Récupération de l'URL de connexion

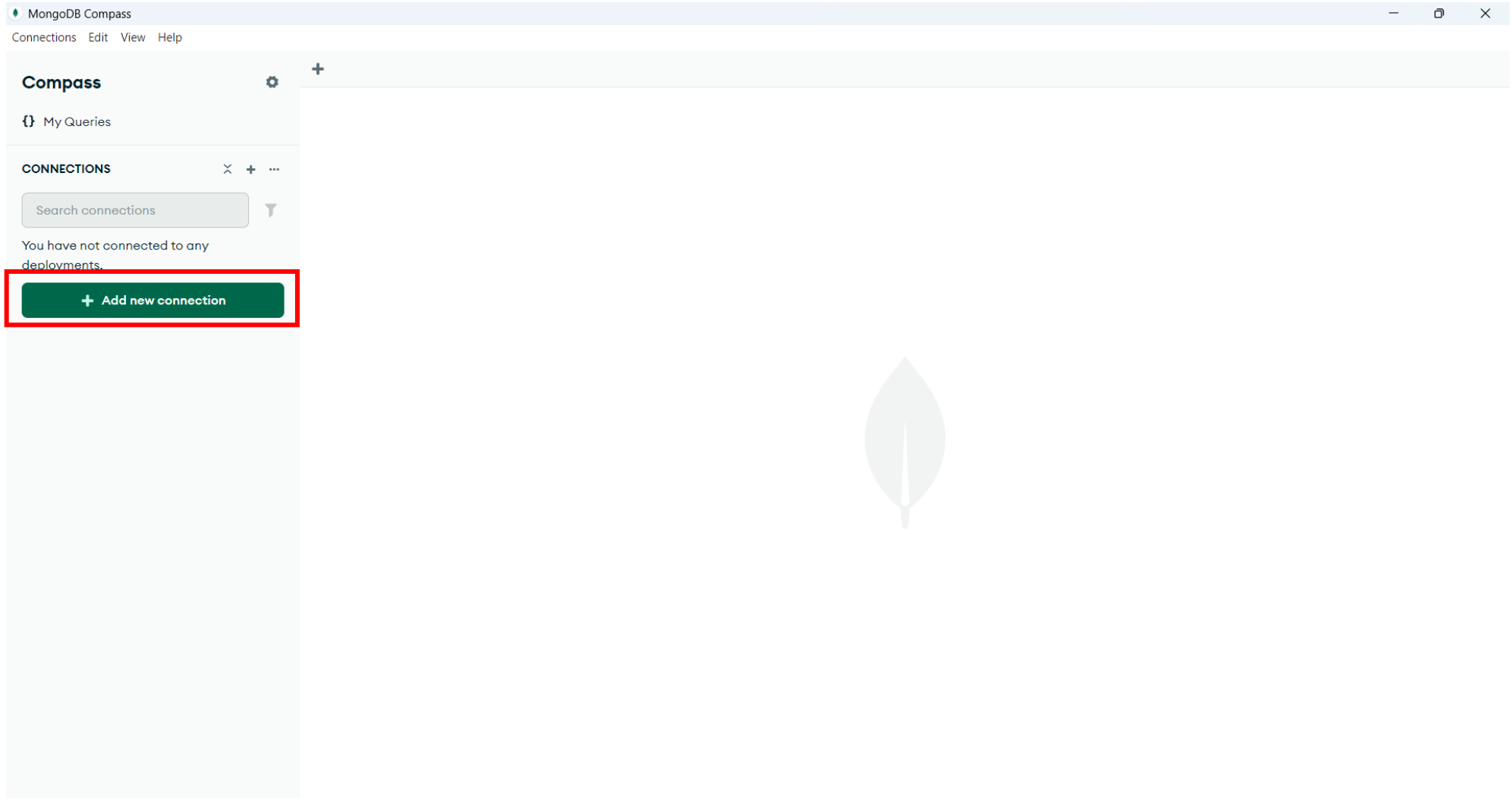


Récupération de l'URL de connexion



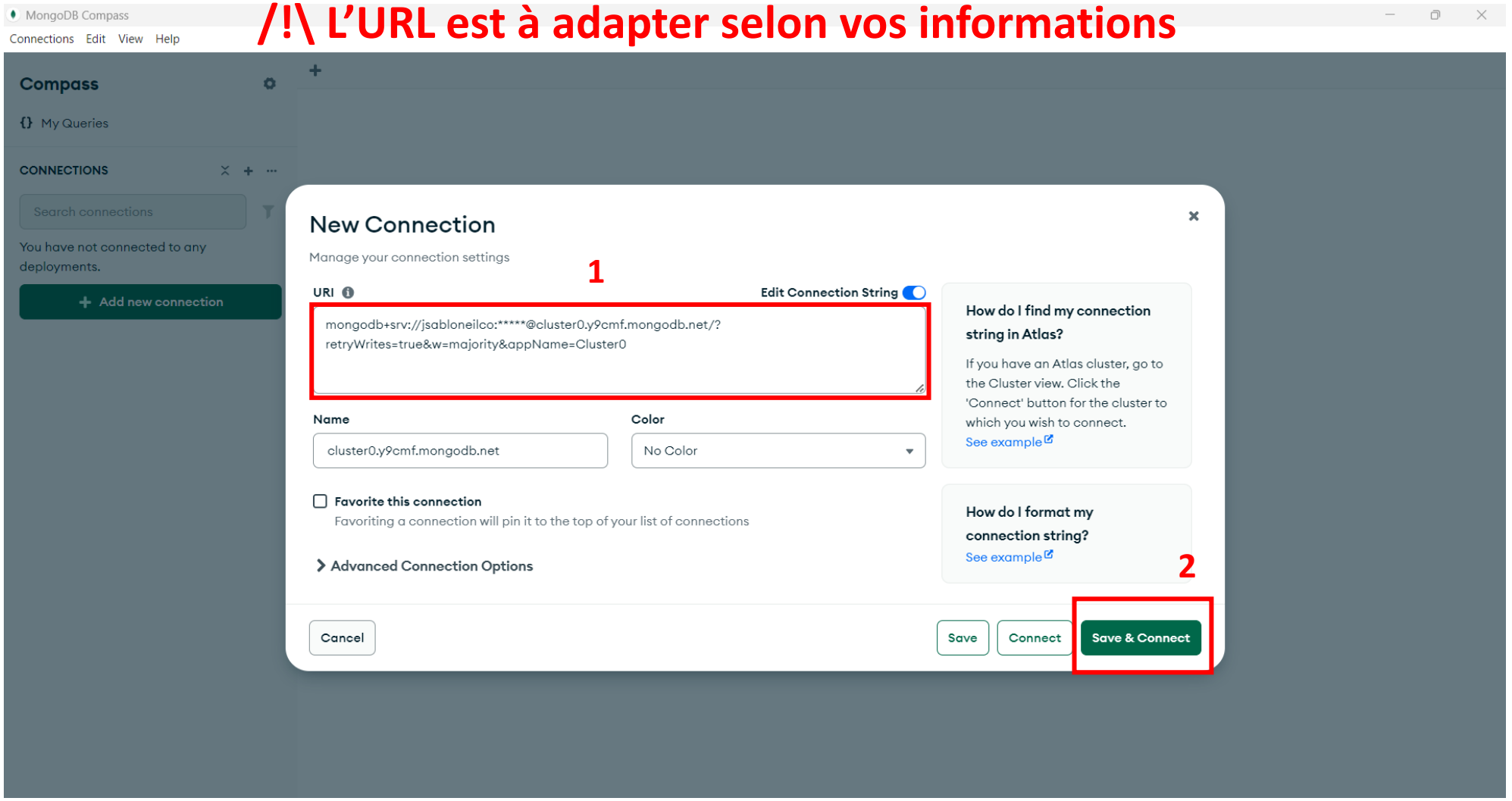
Insertion des données via MongoDB Compass

Démarrer le logiciel `MongoDBCompass` et se connecter à la base de données



Démarrer le logiciel `MongoDBCompass` et se connecter à la base de données

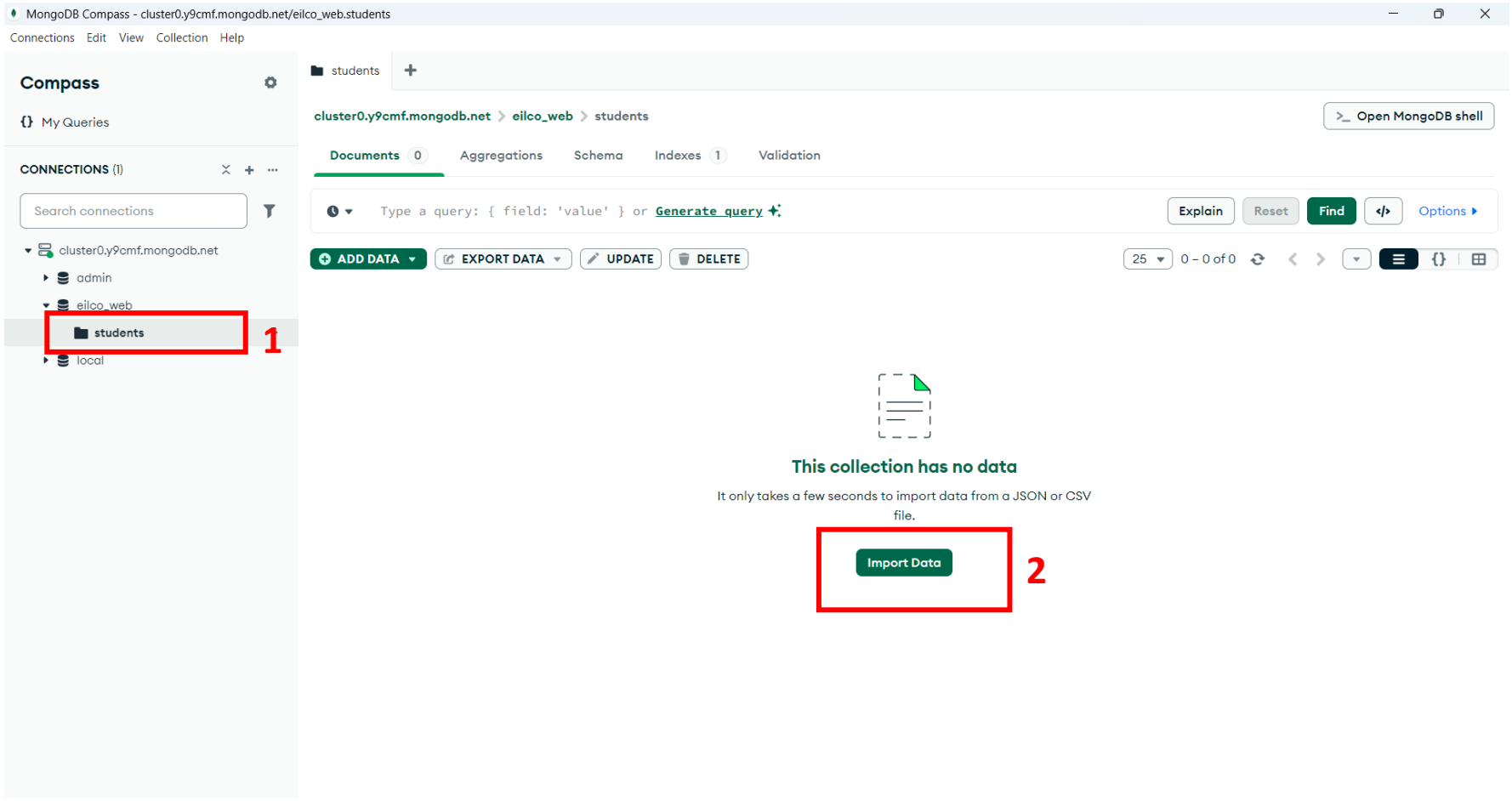
/!\ L'URL est à adapter selon vos informations



Insertion des données via MongoDB Compass

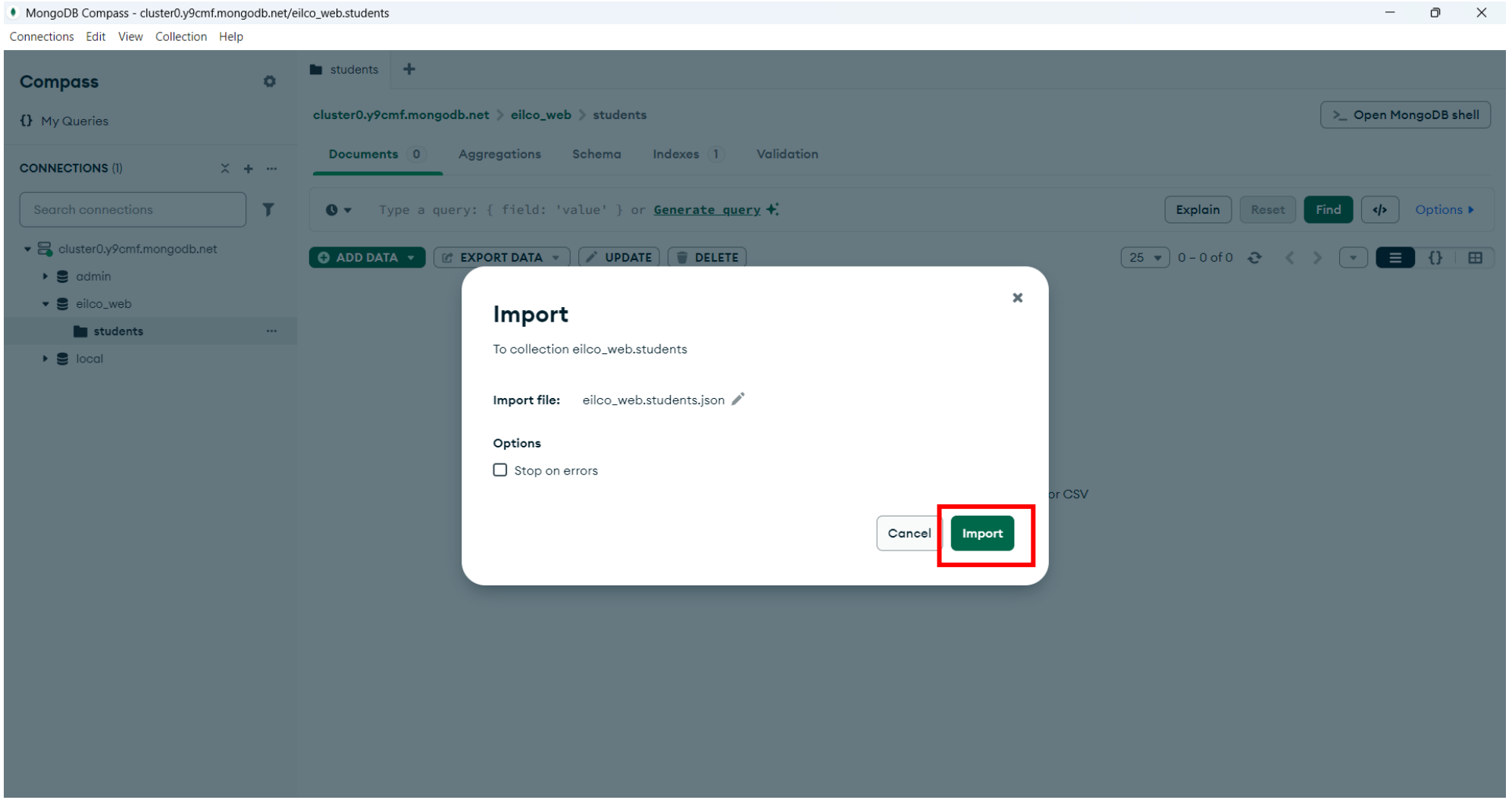
Téléchargez ce fichier de données : [eilco_web.students.json]

Sélectionnez la collection `students` puis « Import Data »



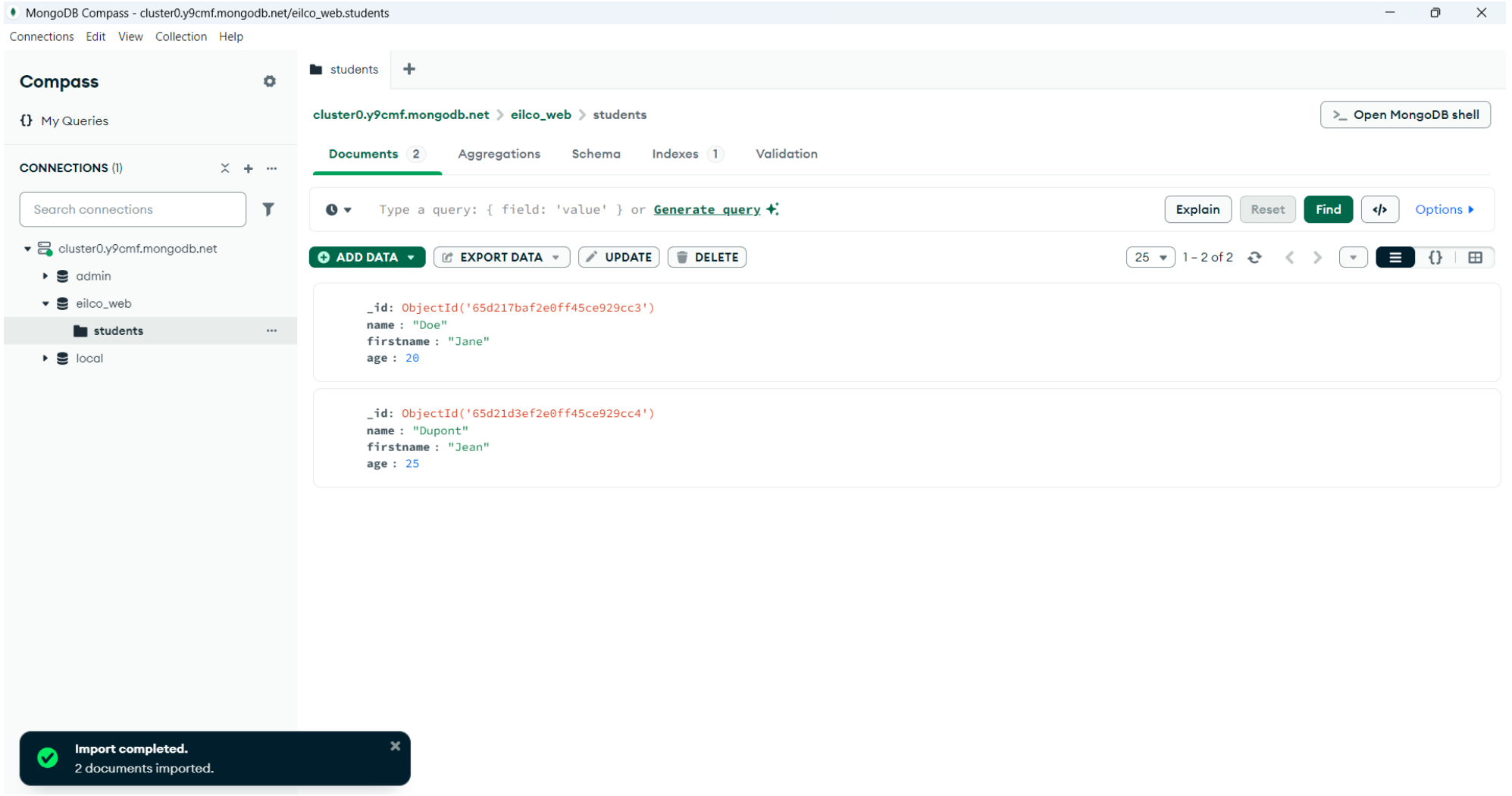
Insertion des données via MongoDB Compass

Sélectionner le fichier `eilco_web.students.json` puis cliquer sur « Import »



Insertion des données via MongoDB Compass

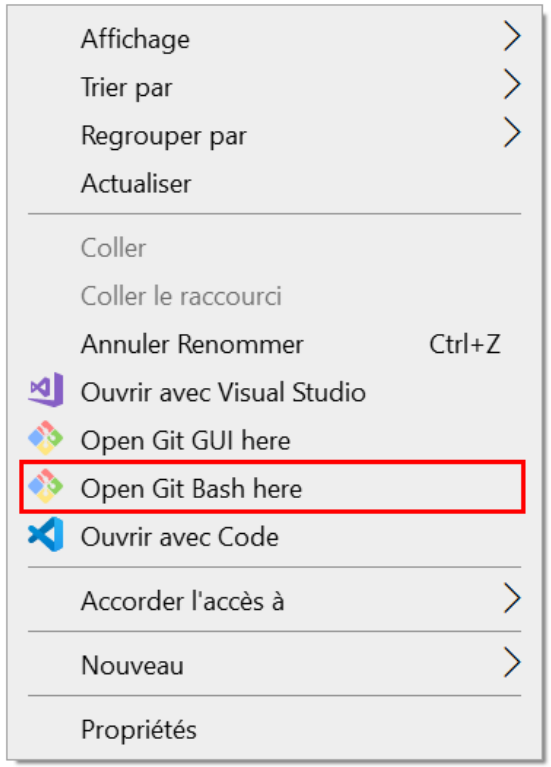
Les données ont bien été importées

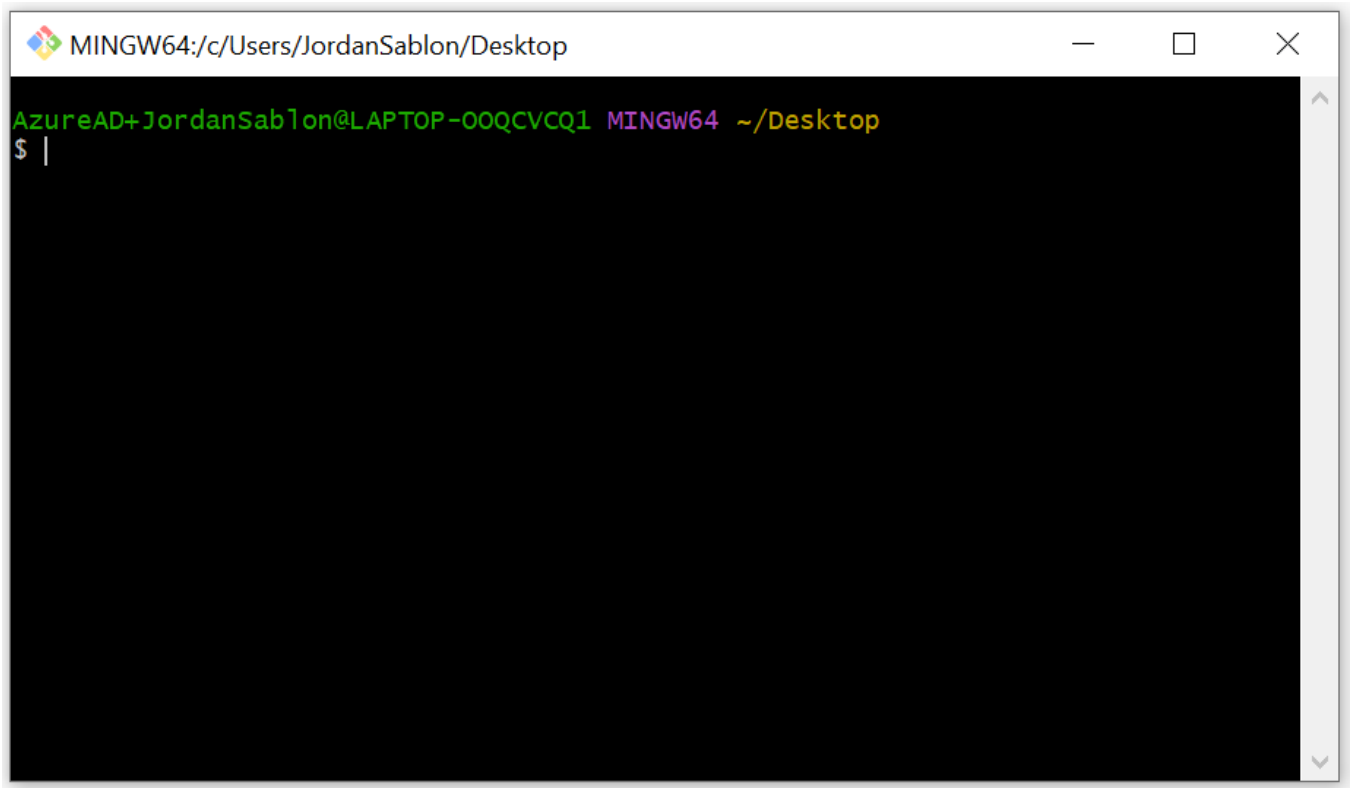


Récupération du code source du projet & installation des dépendances

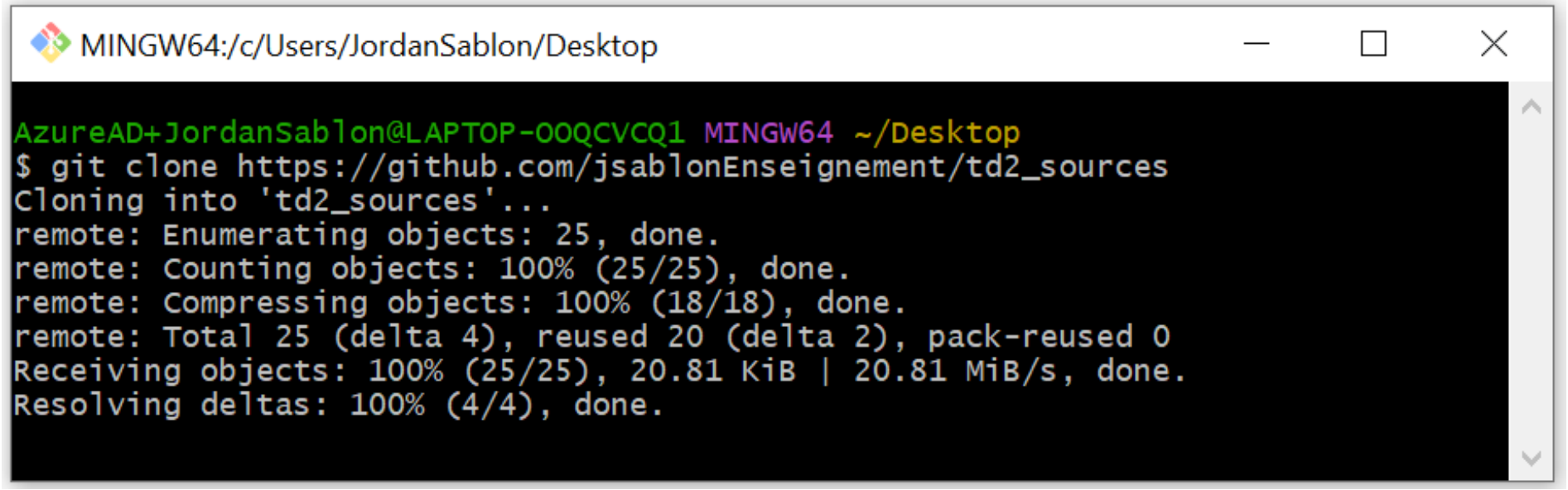
1. Se positionner à l'emplacement de votre choix (où se trouvera le futur dossier du projet)

2. Ouvrir un invité de commande à l'emplacement précédent :



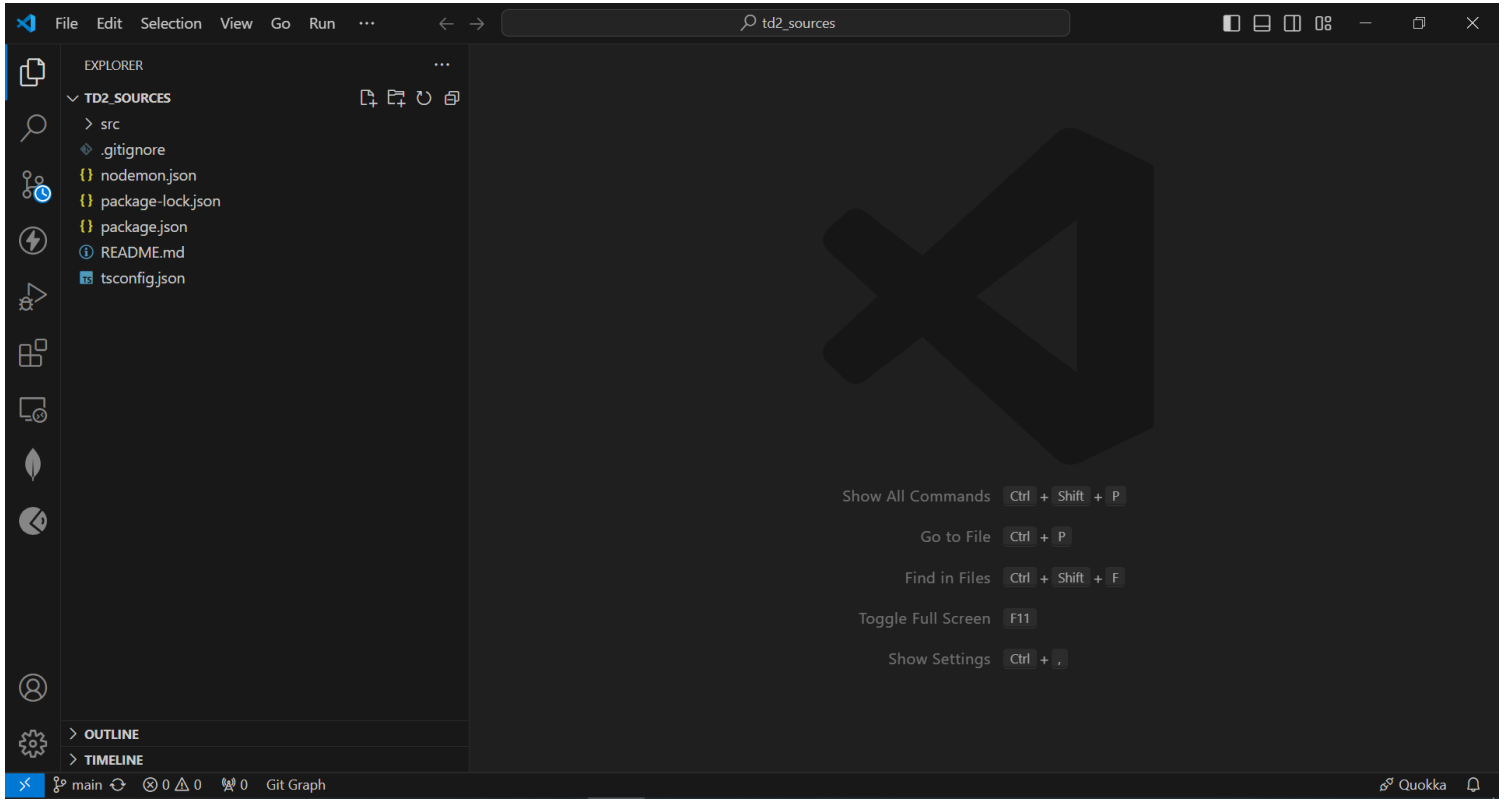


3. Cloner le répertoire Git : https://github.com/jsablonEnseignement/td2_sources



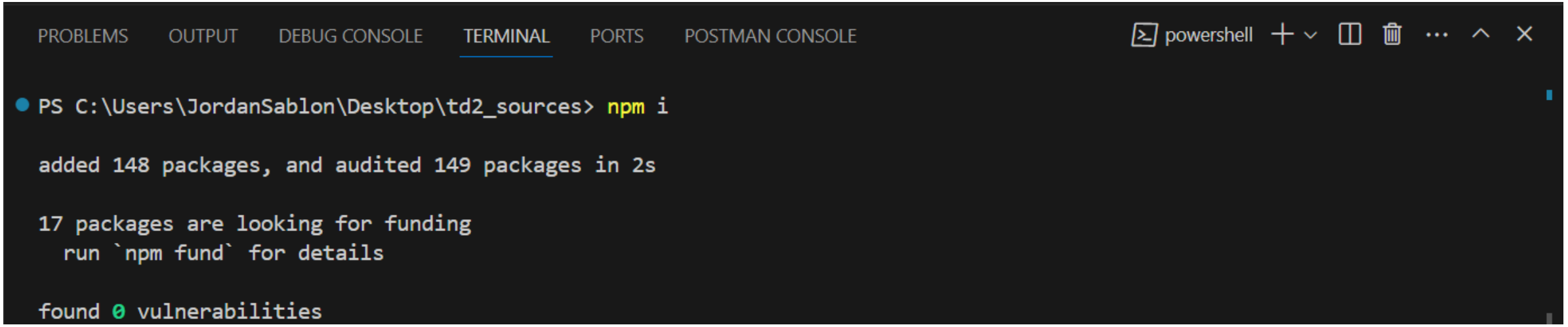
4. Se rendre dans le dossier précédemment cloné :

5. Ouvrir le projet dans VSCode :



Récupération du code source du projet & installation des dépendances

Depuis le terminal de VSCode, exécuter la commande suivante :



Les dépendances nécessaires au bon fonctionnement de l'application viennent d'être installées

Remplacez la chaine de connexion à la base de données dans le fichier **index.ts**

```
mongoose.connect(
  "`mongodb+srv://<username>:<password>@<domain>.mongodb.net/<db_name>?retryWrites=true&w=majority);
const db = mongoose.connection;
```

/!\ L'URL est à adapter selon vos informations

Démarrage du serveur

```
PS C:\Users\JordanSablon\Documents\dev\eilco_web\backend> npm start

> backend@1.0.0 start
> npx tsc && nodemon

[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: ts
[nodemon] starting `ts-node-esm src/index.ts`
Server running at http://127.0.0.1:5000/
Connected successfully
```

Création du modèle

Créez un dossier **`models`** dans le dossier **`/src`**

Créez un fichier **`students.model.ts`** dans le dossier **`/models`**

Implémentez le schéma représentant la collection **`students`** donné à la diapo suivante

Création du modèle

```ts
import { model, Schema, Model, Types, Document } from "mongoose";

const StudentSchema: Schema = new Schema(
  {
    name: {
      type: String,
      required: true,
    },
    firstname: {
      type: String,
      required: true,
    },
    age: {
      type: Number,
      default: 0,
    },
  },
  { versionKey: false }
);

export interface IStudent {
  _id: Types.ObjectId;
  name: String;
  firstname: String;
  age: Number;
}
export interface Student extends Omit<IStudent, "_id">, Document {}

export const StudentModel = model<Student, Model<Student>>(
  "Student",
  StudentSchema,
  "students"
);
```

Récupération de tous les étudiants

Dans le fichier **students.service.ts**, importer le modèle ainsi que l'interface précédemment créés :

```
import { Student, StudentsModel } from "../models/students.model";
```

Modifier la fonction *getStudents* :

```
const getStudents = async () => {
  return await StudentsModel.find();
};
```

ⓘ On applique une fonction **find()** sur le modèle **StudentsModel** afin de récupérer tous les étudiants dans la collection **students** de notre base de données

ⓘ On utilise l'instruction *await* afin de forcer l'attendre du retour de la base de données pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Récupération de tous les étudiants

Dans le fichier **students.controler.ts**, modifier la fonction *getStudents* :

```
export const getStudents = async (req: any, res: any) => {
  const students = await StudentsService.getStudents();
  return res.status(200).json(students);
};
```

ⓘ On utilise l'instruction ***await*** afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Récupération d'un étudiant par son id

Dans le fichier **students.service.ts**, modifier la fonction *getStudent* :

```
const getStudent = async (id: string) => {
  return await StudentsModel.findById(id);
};
```

- ⓘ On applique la fonction **findById()**, avec comme paramètre l'id de l'étudiant, sur le modèle **StudentsModel** afin de récupérer l'étudiant correspondant dans la collection **students** de notre base de données
- ⓘ On utilise l'instruction ***await*** afin de forcer l'attendre du retour de la base de données pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Récupération d'un étudiant par son id

Dans le fichier **students.controller.ts**, modifier la fonction *getStudent* :

```
export const getStudent = async (req: any, res: any) => {
  const { id } = req.params;
  const student = await StudentsService.getStudent(id);
  return res.status(200).json(student);
};
```

ⓘ On utilise l'instruction *await* afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Création d'un étudiant

Dans le fichier **students.service.ts**, modifier la fonction *createStudent* :

```
const createStudent = async (studentToCreate: Student) => {
  const newStudent = new StudentsModel(studentToCreate);
  await newStudent.save();
  return getStudents();
};
```

- ⓘ On crée une nouvelle instance du modèle **StudentsModel** avec les données de notre étudiant puis on applique la fonction *save()* afin d'insérer l'étudiant en base de données

- ⓘ On utilise l'instruction *await* afin de forcer l'attendre du retour de la base de données pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Création d'un étudiant

Dans le fichier **students.controller.ts**, modifier la fonction *createStudent* :

```
export const createStudent = async (req: any, res: any) => {
  const studentToCreate = req.body;
  const students = await StudentsService.createStudent(studentToCreate);
  return res.status(200).json(students);
};
```

ⓘ On utilise l'instruction *await* afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Mise à jour d'un étudiant

Dans le fichier **students.service.ts**, modifier la fonction *updateStudent* :

```
const updateStudent = (id: string, studentToUpdate: Student) => {
  await Student.updateOne(
    {
      _id: new Types.ObjectId(id),
    },
    studentToUpdate
  );
  return getStudents();
};
```

ⓘ On applique une fonction updateOne() sur le modèle **StudentsModel** afin de mettre à jour l'étudiant concerné dans la collection **students** de notre base de données

ⓘ On utilise l'instruction *await* afin de forcer l'attendre du retour de la base de données pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Mise à jour d'un étudiant

Dans le fichier **students.controller.ts**, modifier la fonction *updateStudent* :

```
export const updateStudent = async (req: any, res: any) => {
  const { id } = req.params;
  const studentUpdated = await StudentsService.updateStudent(
    id,
    req.body
  );
  return res.status(200).json(studentUpdated);
};
```

ⓘ On utilise l'instruction ***await*** afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Suppression d'un étudiant

Dans le fichier **students.service.ts**, modifier la fonction *deleteStudent* :

```
const deleteStudent = async (id: string) => {
  await StudentsModel.deleteOne({ _id: new Types.ObjectId(id) });
};
```

- ⓘ On applique une fonction deleteOne() sur le modèle **StudentsModel** afin de supprimer l'étudiant correspondant de la collection **students** de notre base de données
- ⓘ On utilise l'instruction *await* afin de forcer l'attendre du retour de la base de données pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)
- ⓘ On fait le choix de ne pas retourner de données

Suppression d'un étudiant

Dans le fichier **students.controller.ts**, modifier la fonction *deleteStudent* :

```
export const deleteStudent = async (req: any, res: any) => {
  const { id } = req.params;
  await StudentsService.deleteStudent(id);
  return res
    .status(200)
    .json(`Student with id ${id} succesfully deleted`);
};
```

ⓘ On utilise l'instruction ***await*** afin de forcer l'attendre l'exécution de notre service pour retourner les étudiants. On précise alors que notre fonction est asynchrone (*async*)

Utilisation des variables d'environnement

Objectifs :

- éviter d'envoyer des informations sensibles sur votre repo git
- spécifier une configuration variable selon les environnements tout en conservant un code générique

Installez la dépendance **`dotenv`**

```
npm install dotenv
```

Importez **`dotenv`** dans le fichier **`index.ts`**

```
import dotenv from "dotenv";
```

Ajoutez cette ligne de code en début de fichier (après les imports)

```
dotenv.config();
```

Créez un fichier **.env** à la racine du projet (au même niveau que le **package.json**) qui contiendra les informations suivantes :

```
DB_USERNAME=your_username
DB_PASSWORD=your_password
DB_DOMAIN=your_domain.mongodb.net
DB_NAME=your_db_name
```

/!\ Adapter les valeurs de droite ci-dessus en fonction de vos informations de connexion à la base de données

Modifiez votre ligne de connexion à la base de données dans le fichier **index.ts** pour utiliser les variables ci-dessus au lieu des valeurs :

```
mongoose.connect(
  `mongodb+srv://${process.env.DB_USERNAME}:${process.env.DB_PASSWORD}@${proce
ss.env.DB_DOMAIN}/${process.env.DB_NAME}?retryWrites=true&w=majority`
)
```