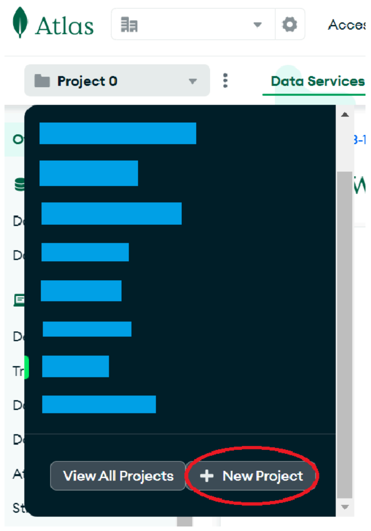
Next.js: Persistence with MongoDB

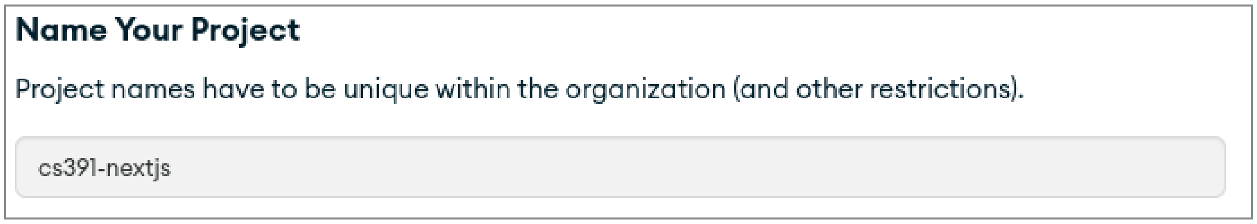
This week we will connect our project to MongoDB, allowing changes to persist in a database. This means changes will continue to exist after refresh and can be seen by others users.

Task 1: Setting up MongoDB

1. Visit https://mongodb.com and create an account or sign in to your existing account. It does not matter which method you choose to sign in with, any will work
2. Select the dropdown in the top left and click **New Project**



3. Then give your project a name and click next. You do not need to add any tags.

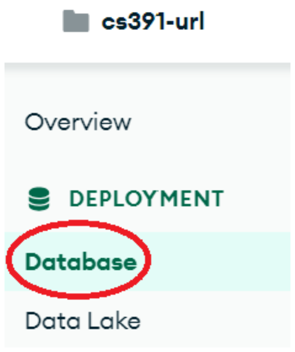


Name Your Project

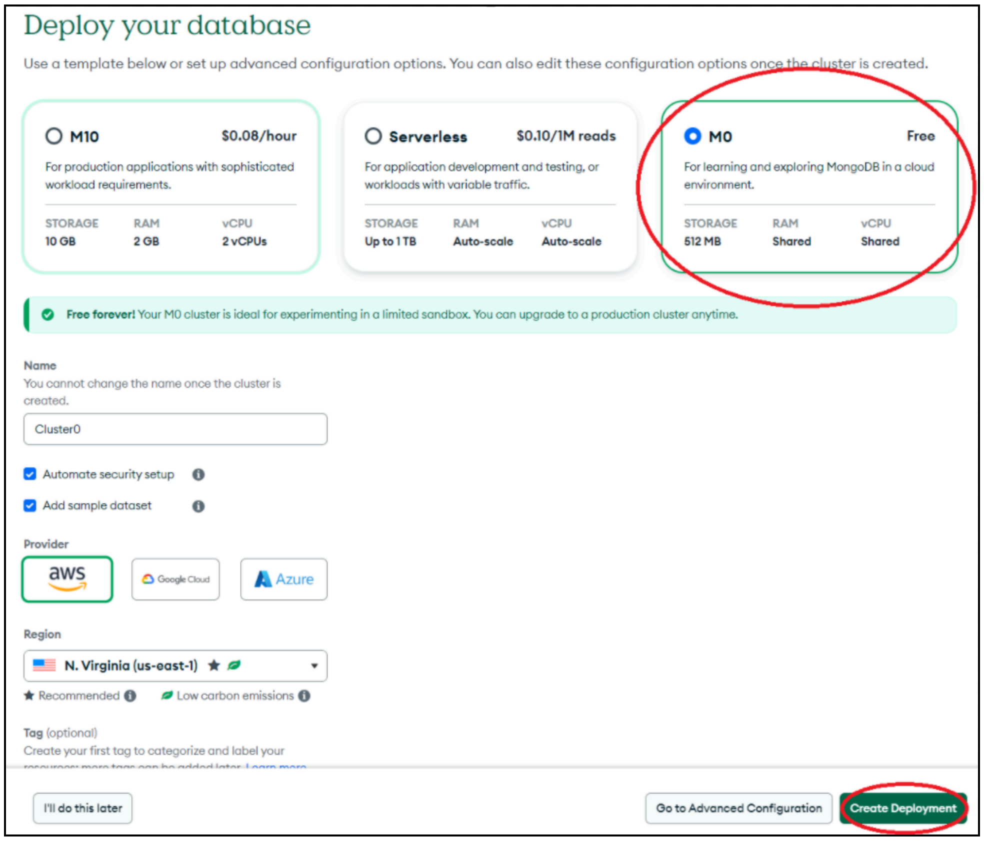
Project names have to be unique within the organization (and other restrictions).

cs391-nextjs

4. On the next page, just click **Create Project**
5. Then select **Database** on the left hand side. Then click **Build a Database**



6. Select **M0** then **Create Deployment**. This option should be free



Deploy your database

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

○ M10	$0.08/hour		
For production applications with sophisticated workload requirements.			
STORAGE 10 GB	RAM 2 GB	vCPU 2 vCPUs	
○ Serverless	$0.10/1M reads		
For application development and testing, or workloads with variable traffic.			
STORAGE Up to 1 TB	RAM Auto-scale	vCPU Auto-scale	
● M0	Free		
For learning and exploring MongoDB in a cloud environment.			
STORAGE 512 MB	RAM Shared	vCPU Shared	

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

Name
You cannot change the name once the cluster is created.
Cluster0

✓ Automate security setup
✓ Add sample dataset

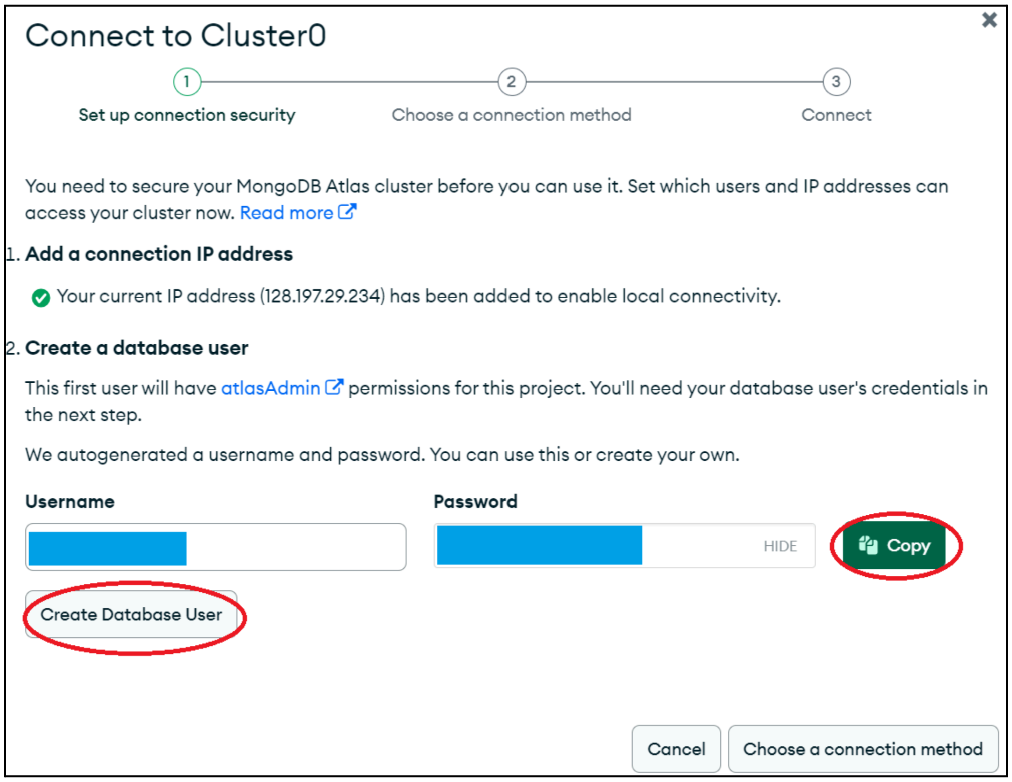
Provider
aws | Google Cloud | Azure

Region
N. Virginia (us-east-1) ★

★ Recommended   Low carbon emissions

Tag (optional)
Create your first tag to categorize and label your

7. Copy your username and password. Then click **Create Database User**



Connect to Cluster0

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

1. **Add a connection IP address**

✓ Your current IP address (128.197.29.234) has been added to enable local connectivity.

2. **Create a database user**

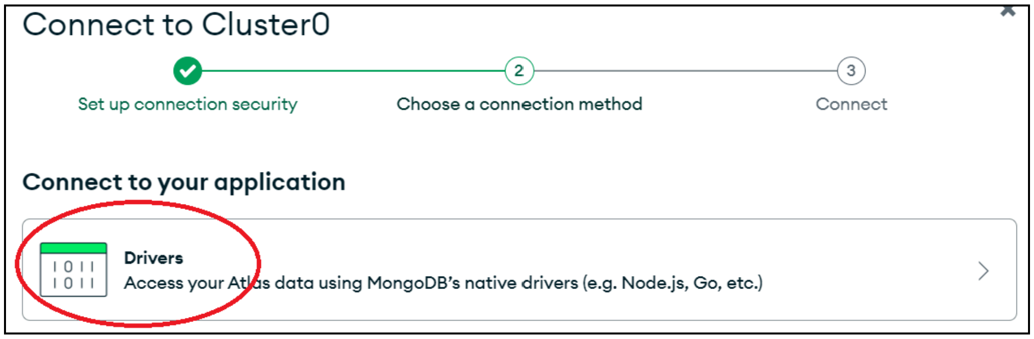
This first user will have atlasAdmin permissions for this project. You'll need your database user's credentials in the next step.

We autogenerated a username and password. You can use this or create your own.

Username [redacted]

Password [redacted] HIDE

8. Click **Choose a connection method**. Then click **Drivers**



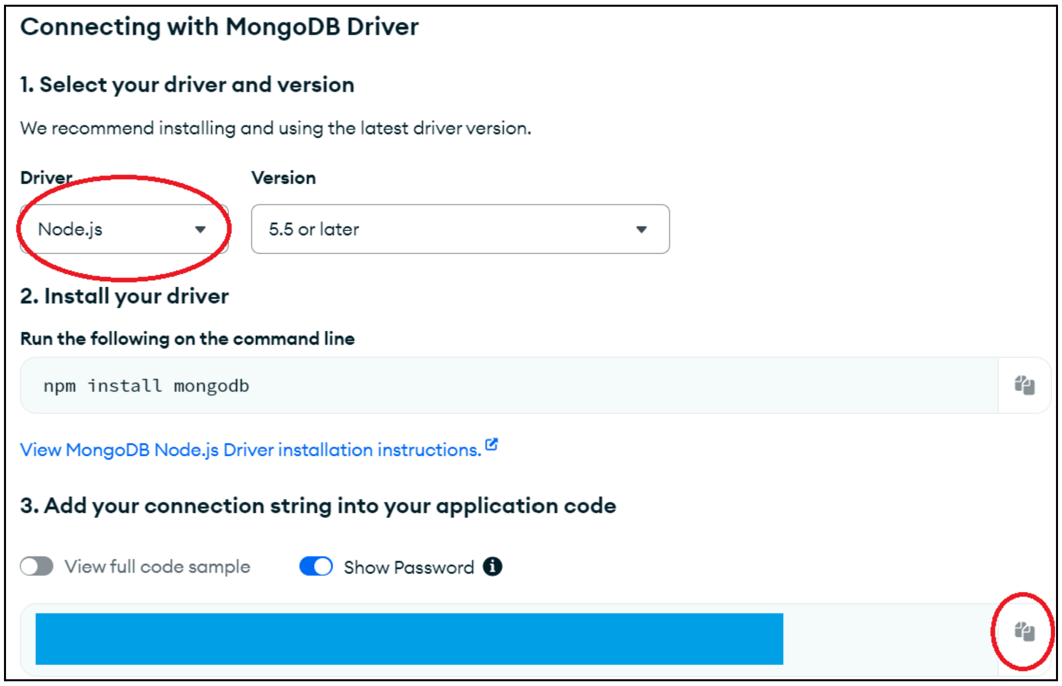
Connect to Cluster0

Set up connection security — Choose a connection method — Connect

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

9. Select **Node**, then copy your connection string



Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver: Node.js
Version: 5.5 or later

2. Install your driver

Run the following on the command line

```
npm install mongodb
```

View MongoDB Node.js Driver installation instructions.

3. Add your connection string into your application code

View full code sample — Show Password

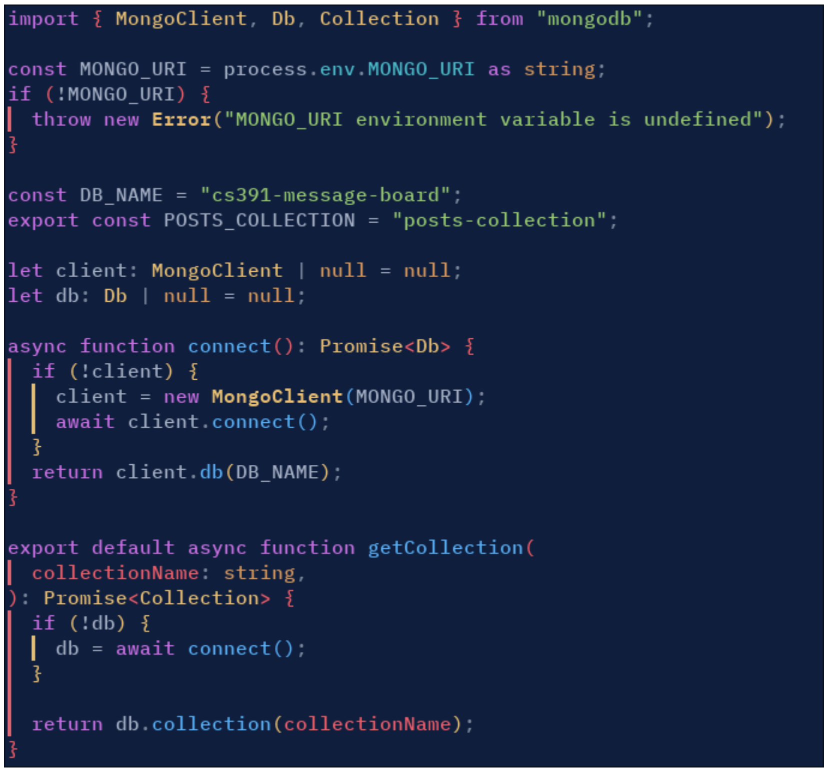
[redacted]

10. Make a **.env.local** file at the root of your project. Then create a variable called **MONGO_URI** and set it to your connection string. Make sure your username and password are both included in the string
    a. In the connection string, you will need to replace **<db_username>** with the username you chose and you will need to replace **<db_password>** with the corresponding password
    b. By default, **.env.local** file should be included in the **.gitignore** file created by Next.js. Ensure this is the case, you don't want to accidentally leak your credentials
11. Then go to the next page and click **Done**
12. If sample data was loaded into your database, you may remove it. This step is not required.

Task 2: Connect to MongoDB

1. Install the MongoDB package using the command **npm install mongodb**

2. Create a file called **db.ts** at the root of your project. This is where we will connect to MongoDB

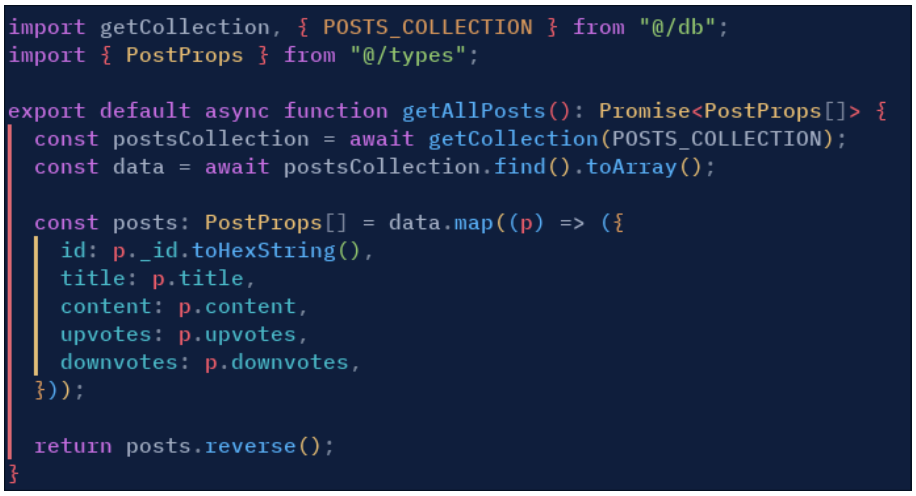


```
import { MongoClient, Db, Collection } from "mongodb";

const MONGO_URI = process.env.MONGO_URI as string;
if (!MONGO_URI) {
  throw new Error("MONGO_URI environment variable is undefined");
}

const DB_NAME = "cs391-message-board";
export const POSTS_COLLECTION = "posts-collection";

let client: MongoClient | null = null;
let db: Db | null = null;

async function connect(): Promise<Db> {
  if (!client) {
    client = new MongoClient(MONGO_URI);
    await client.connect();
  }
  return client.db(DB_NAME);
}

export default async function getCollection(
  collectionName: string,
): Promise<Collection> {
  if (!db) {
    db = await connect();
  }

  return db.collection(collectionName);
}
```

   a. Here are some resources on MongoDB databases and collections
      i. https://www.mongodb.com/docs/manual/core/databases-and-collections/
      ii. https://www.mongodb.com/docs/compass/current/collections/

Task 3: Read from MongoDB

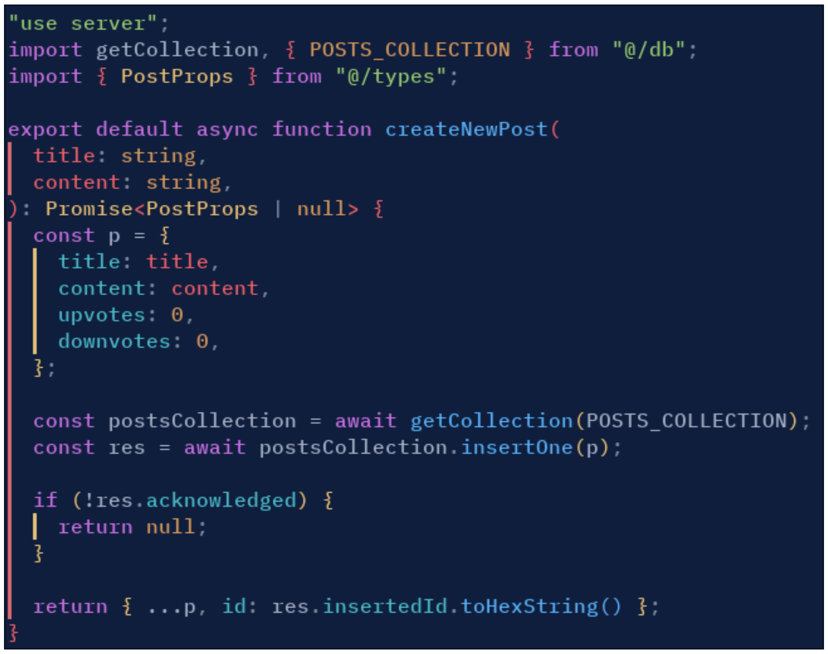
1. Edit the function **getAllPosts()** to retrieve entries from the database
   a. Here we use the find() and toArray() methods to retrieve an array of all posts stored in this database collection



```
import getCollection, { POSTS_COLLECTION } from "@/db";
import { PostProps } from "@/types";

export default async function getAllPosts(): Promise<PostProps[]> {
  const postsCollection = await getCollection(POSTS_COLLECTION);
  const data = await postsCollection.find().toArray();

  const posts: PostProps[] = data.map((p) => ({
    id: p._id.toHexString(),
    title: p.title,
    content: p.content,
    upvotes: p.upvotes,
    downvotes: p.downvotes,
  }));

  return posts.reverse();
}
```

Task 4: Write to MongoDB

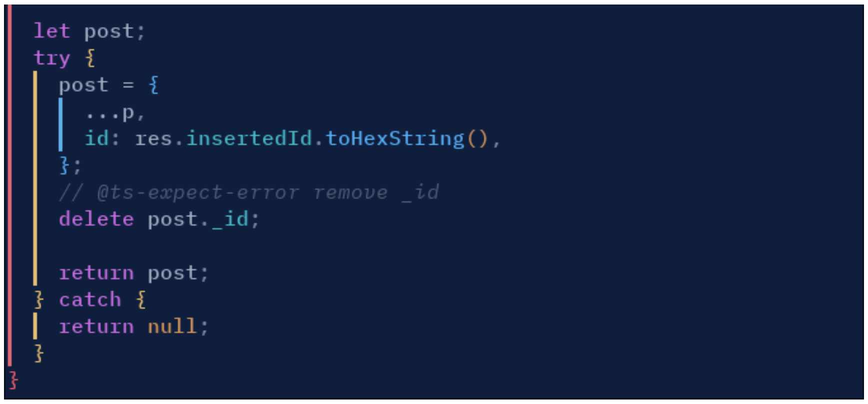
1. Edit the function **createNewPost()** to insert a new post into the proper database collection
   a. Here we use the insertOne() method to insert a new entry into the collection
   b. We then check if that operation was successful and return the new post object if it was



```
"use server";
import getCollection, { POSTS_COLLECTION } from "@/db";
import { PostProps } from "@/types";

export default async function createNewPost(
  title: string,
  content: string,
): Promise<PostProps | null> {
  const p = {
    title: title,
    content: content,
    upvotes: 0,
    downvotes: 0,
  };

  const postsCollection = await getCollection(POSTS_COLLECTION);
  const res = await postsCollection.insertOne(p);

  if (!res.acknowledged) {
    return null;
  }

  return { ...p, id: res.insertedId.toHexString() };
}
```

Task 5 (Optional): A small bug

1. Not passing plain objects
   a. You may see an error stating "Only plain objects can be passed to Client Components from Server Components". This can be remedied by removing the **_id** attribute from the post returned by the **createNewPost()** function. However, it is likely that your IDE will complain about this. Use a @ts-expect-error to make it go away



```
  let post;
  try {
    post = {
      ...p,
      id: res.insertedId.toHexString(),
    };
    // @ts-expect-error remove _id
    delete post._id;

    return post;
  } catch {
    return null;
  }
}
```