

[Guides](#)[Changelog](#)[Blog](#)[← Back to Guides](#)

By Humberto Leal

Published 30th November 2021

DATABASES

Connecting to a MongoDB database using Node.js

Node.js

Databases

Development

NoSQL

MongoDB is a NoSQL document-based database. It supports different deployments for high-availability and scalability use cases. This guide introduces you to connecting your Node.js application to a MongoDB database and how to read and write data.

Goals

At the end of this guide you will be able to create a Node.js project, setup a connection to a MongoDB database and read and write some data.

Prerequisites

- Your local machine with Node.js & npm installed <https://nodejs.org/>
- Create a new directory and initialize an empty Node.js project with `npm init`
- A running instance of MongoDB with TLS configured

Project structure

```
node-with-mongodb/  
├── package.json  
├── .env  
└── mongodb.js
```

The full source code used in this guide can be found in [this git repository](#).

Node.js + MongoDB

In this guide, we will be using the official Node.js MongoDB module to connect to our database. We'll also use the `dotenv` package to load environment variables specified in the `.env` file when the script is executed. They can be installed using npm:

```
npm install mongodb  
npm install dotenv
```

All the code examples will be part of the `mongodb.js` file in the project directory.

Environment variables `.env` file

The `.env` file will contain the URI connection string. This contains the user, password, host address as well as database and extra connection options for connecting to your MongoDB database.

```
MONGO_URI='mongo+srv://<user>:<pass>@<host>:<port>/<database>?<connection options>'
```

Once the `.env` file is ready, in the `mongodb.js` script we can load the contents of that file as environment variables:

```
1 require('dotenv').config()
```

This will allow us to use `process.env.MONGO_URI` in our scripts.

Connecting to the database with TLS

Connecting to your database with TLS encrypts all network data between the MongoDB server and your client. This also helps to ensure the client is connected to the intended server.

The Node.js MongoDB library supports this setting. The simplest way to connect to your database is by using the URI with all the required data: user, password, database, host, port, and other options such as replica set, TLS, or authSource.

The URI should have the host and all the credentials required for connecting to the database. In case of using TLS, an extra parameter `tls:` `true` can be provided to the MongoClient constructor.

```
1 const client = new MongoClient(uri, {  
2   tls: true,  
3 })
```

However, if you use a connection URI with `mongo+srv` prefix, `tls` will be automatically enabled, so the `tls: true` option can be omitted.

The following snippet shows the client configuration and connection to the database. It also queries the database information and prints the role of the MongoDB node.

```
1  require('dotenv').config()
2  const { MongoClient } = require('mongodb');
3  const uri = process.env.MONGO_URI;
4  const client = new MongoClient(uri);
5
6  (async () => {
7    try {
8      await client.connect();
9      const dbRole = await client.db().command({ hello: 1 });
10     console.log(
11       `Role of database - Host: ${dbRole.me}  Is primary: ${dbRole.isWrit
12     );
13     await client.close();
14   } catch (e) {
15     console.log('Error: ', e.message);
16   }
17 })();
```

Reading and writing data

Now that we're connected to our database, we can proceed to write and read data.

In order to do this, we have to use the MongoClient we created in previous steps and access a collection of documents called 'movies':

```
1  await client.connect();
2
3  const moviesCollection = await client.db().collection('movies');
4
5  console.log('Collection name: ', moviesCollection.name);
```

If the collection doesn't exist it will be automatically created for us. The next step is to add document to the `movies` collection:

```
1  const { insertedId } = await moviesCollection.insertOne({ name: 'Spider-
```

The `insertedId` is an `ObjectId` that represents the ID of the new document we just inserted into the collection. We can then use that same `ObjectId` to query the collection and fetch the document:

```
1 const document = await moviesCollection.findOne({ _id: insertedId });
2 console.log('Document from db: ', document);
```

We can also fetch the number of documents part of a collection by calling the `countDocuments()` method as follows:

```
1 console.log('Number of documents: ', await moviesCollection.countDocument
```

Full example

Now we can use the full script as follows, with all of the logic written in previous steps. This involves: Connecting to our database using TLS; adding entries into 'movies' collection; querying a document using `ObjectId` and getting the number of documents on the collection.

The full script would be as follows:

```
require('dotenv').config();
const { MongoClient } = require('mongodb');
const uri = process.env.MONGO_URI;

const client = new MongoClient(uri);
(async () => {
  try {
    await client.connect();

    const dbRole = await client.db().command({ hello: 1 });

    console.log(
      `Role of database - Host: ${dbRole.me} Is primary: ${dbRole.isWriteable}
    );

    // Accessing 'movies' collection object
    const moviesCollection = await client
      .db()
      .collection('movies');

    console.log('Collection name: ', moviesCollection.collectionName);
```

```
24 // Inserting new document into the 'movies' collection
25 const result = await moviesCollection.insertOne({ name: 'Spider-Man:
26
27 const { insertedId } = result;
28 console.log('Result: ', insertedId);
29
30 // Fetching a document based on ObjectId
31 const document = await moviesCollection.findOne({ _id: insertedId });
32 console.log('Document from db: ', document)
33
34 // Getting the number of documents in the collection.
35 console.log(
36   'Number of documents: ',
37   await moviesCollection.estimatedDocumentCount()
38 );
39
40 await client.close();
41 } catch (e) {
42   console.log('Error: ', e.message);
43 }
44 })();
```

Optional: Validating Server Certificates Through CAs.

In scenarios where the certificate served by the MongoDB database is not issued by a well-known certificate authority, connection using TLS will fail. As a consequence, the user must provide the certificate authority to verify the server certificate. This can be done through the `ca` option on the `MongoClient` class.

The following snippet shows how to provide a certificate authority for connecting with TLS to your MongoDB database. The file is read from the path provided to the `readFileSync` method.

```
1 require('dotenv').config();
2 const { MongoClient } = require('mongodb');
3 const fs = require('fs');
4
5 const uri = process.env.MONGO_URI;
6
7 const client = new MongoClient(uri, {
8   tls: true,
9   ca: [fs.readFileSync('<path to certificate authority>')]
10 });
```

Then you can connect to your database as usual:

```
1 (async () => {  
2   await client.connect();  
3 })();
```

[Sign up](#)

In this how-to guide, we have shown how to use Node.js to connect to a MongoDB instance and how to read and write data. The full source code used in this guide can be found in [this git repository](#).

In the first step, we create an .env file which contains the URI connection string to MongoDB. These variables are then used to connect to a MongoDB instance. In the final step, we showed how to read and write data from our database.

Using Northflank to connect Node.js to MongoDB for free

Northflank allows you to spin up a MongoDB database and a Node.js service within minutes. Sign up for a Northflank account and create a free project to get started.

- Multiple read and write replicas
- Observe & monitor with real-time metrics & logs
- Low latency and high performance
- Backup, restore and fork databases
- Private and optional public load balancing as well as Northflank local proxy

[Get started now](#)

SHARE THIS ARTICLE WITH YOUR NETWORK



RELATED ARTICLES



+



Northflank

[DEPLOY ON NORTHFLANK](#)

Deploy Grafana on Northflank

Grafana allows you to query, visualize, alert on and understand your metrics. In this guide, we'll walk through deploying Grafana on Northflank.

16th September 2022 • Thomas Smyth

[Databases](#)[NoSQL](#)[SQL](#)[Docker Images](#)[Apps](#)

+



Northflank

[DEPLOY ON NORTHFLANK](#)

Deploy tRPC on Northflank

tRPC allows you to easily build & consume fully typesafe APIs. In this guide, we walk through deploying a tRPC app on Northflank.

31st August 2022 • Tom Snelling

Node.js

Databases

TypeScript



DATABASES

Use a MinIO S3 bucket on Northflank






MinIO is an excellent option if you require high-performance object storage with an S3-compatible API

18th April 2023 • Daniel Cosby


Databases

S3

NEXT ARTICLES

-  [Connecting to a Redis database using Node.js](#)
-  [Connecting to a MinIO database using Node.js](#)
-  [Connecting to a PostgreSQL database using Python](#)
-  [Deploying Metabase with PostgreSQL](#)
-  [Deploying Retool with PostgreSQL](#)



 All systems operational

© 2023 Northflank Ltd. All rights reserved.

Features

[Overview](#)

[Platform](#)

[Build](#)

[Run](#)

[Release](#)

[Scale](#)

[Observe](#)

[Databases](#)

[Templates](#)

[Managed Cloud](#)

[Bring Your Cloud](#)

Product

[Pricing](#)

[Schedule a demo](#)

[Log in](#)

[Sign up](#)

Company

[About Northflank](#)

[Careers](#)

[Terms of Service](#)

[Privacy Policy](#)

[Contact us](#)

Resources

[Changelog](#)

[Blog](#)

[Guides & tutorials](#)

[Status](#)

[Documentation](#)

[API reference](#)


Registered office

Company 11918540
20-22 Wenlock Road,
London, England, N1 7GU

Get in touch

✉ contact@northflank.com

 [LinkedIn](#)

 [Twitter @northflank](#)