Deploying a Node app

There are different ways to deploy. If there's just Node (MongoDB is a different story), it's relatively simple. For testing/prototyping Glitch or Google Cloud Platform are two interesting ones you can try.

Netlify (https://netlify.com)

- https://www.netlify.com/blog/2018/09/13/how-to-run-express.js-apps-with-netlify-functions/

Vercel (https://vercel.com)

- https://dev.to/adafycheng/deploy-nodejs-application-to-vercel-in-5-minutes-171m
- https://vercel.com/guides/using-express-with-vercel

Note that there are performance implications when using serverless functions (e.g. with Netlify and Vercel) for deploying an Express.js app but these are fine for simple one-offs as a prototype. The second link above goes over some of the issues.

Glitch (https://glitch.com/)

- For prototyping or just playing around, there's an option to create a small anonymous app (app expires in a short time).
- You can also sign up for non-anonymous apps though there are limits for the free tier.

Google Cloud Platform

- This is paid but when you first login (you can just use your regular Google account if you've got a Gmail account), if you haven't used Google Cloud Platform before, you should be able to get free credits ($300 worth as of this writing).
- Google has a guide for building and deploying a Node.js app.

Paid options

Traditional hosting (shared)

If using shared hosting already, check to see if your host supports Node apps. If using a Linux-based shared host, you'll probably be using cPanel as the admin interface. If Node is supported, there's probably a one-click installer for you to set up your app. Just follow the instructions for your host. Here's a good read:

https://medium.com/@pampas93/host-your-node-js-app-on-shared-hosting-go-beyond-localhost-73ab923e6691

Cloud hosting (e.g. Digital Ocean)

For cloud servers, Digital Ocean is a good one to try. For Digital Ocean you get a droplet which is a virtual machine but acts like a separate server. This means that you need to set it up but it does give you more control. **Digital Ocean is paid but you can start with the cheaper tier ($5/mo).**

More recently, Digital Ocean also offers a one-click setup to make it easier for you.

- Create and deploy Node app to Digital Ocean (older documentation but may be useful): https://www.digitalocean.com/community/tutorials/deploying-a-node-app-to-digital-ocean
- Create a one-click Node.js app: https://marketplace.digitalocean.com/apps/nodejs
- Managed MongoDB (more expensive): https://www.digitalocean.com/blog/introducing-digitalocean-managed-mongodb/

MongoDB Atlas

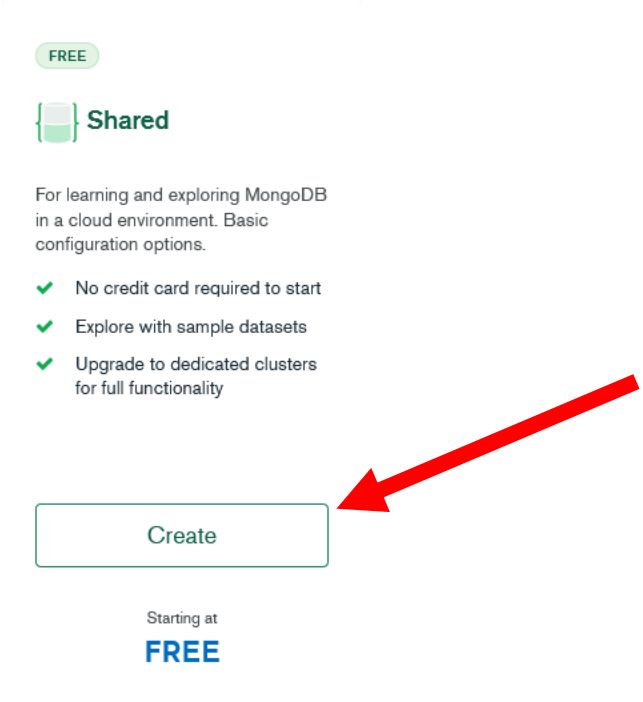
You're probably more familiar with MySQL and phpMyAdmin. For phpMyAdmin you can export your database by selecting the database to export then going to the **Export** tab and running the export to file download.

For MongoDB we'll be using MongoDB Compass and MongoDB Atlas for the online server. For testing/prototyping purposes we can use the free tier. (For us, with little projects it's good enough.)

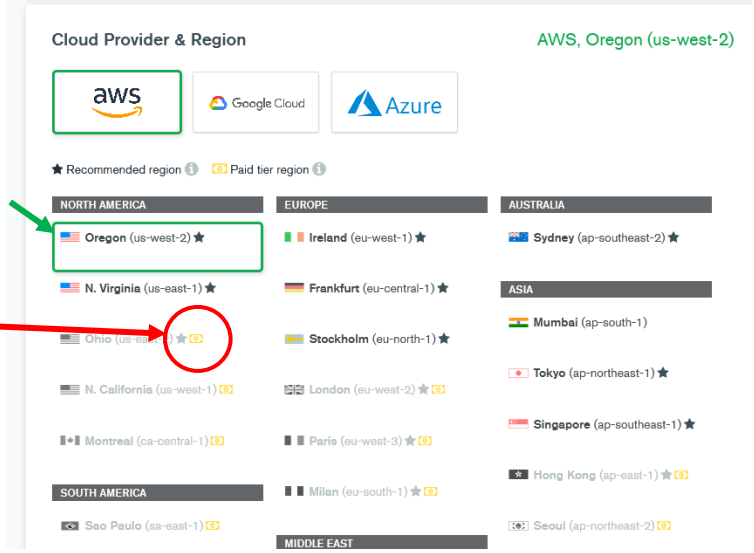
Create Cluster and DB on MongoDB Atlas (https://www.mongodb.com/atlas)

You can have **only one free cluster per project**.

1. Create an account and login.
2. There should be a project created by default (if not, create a project). Click to build a database.
3. Select to create a Shared (free) database.



4. When choosing the cluster location, choose carefully and avoid the ones that say they're paid tier regions only.



5. Click **Create Cluster** at bottom of the page.
6. Once the cluster is finished provisioning, you should be asked to secure your connection. If you can't see this, click on **Quickstart** under SECURITY in the sidebar.
7. Set up a user name and password for an admin user and add your local IP (this will allow you to access from your local environment). You can also choose "Allow Access From Anywhere" to make your data readable from anywhere.

> *Configuring IP access*
>
> Later, once you've deployed your project, if you're using MongoDB, you'll need to add your site server's IP to the allowed IP addresses list.

8. Create the DB you want to migrate. Note that you can't migrate everything all at once using Atlas live to a free tier cluster, so we'll duplicate the DB and collection names. (It's easier if you start with a live MongoDB database on MongoDB Atlas.)
    a. Click on **DEPLOYMENT > Databases** in the sidebar.
    b. Click on the **Browse Collections** button.
    c. Click the **+ Create Database** button and create the "testdb" database and "menuLinks" collection.
9. Once that's done, you can create another user who will only have read/write privileges for your DB.
    a. Click on **Database Access**.
    b. Click on **+ Add New Database User**.
    c. Use "testdbuser" as the username and you can auto-generate the password (don't forget to copy/paste this somewhere).
    d. For the user privileges, expand "Specific Privileges" and click **Add Specific Privilege**.
    e. Select "readWrite" for the role and type "testdb" for the database.
    f. Under "Built-in Role" click on the garbage can to delete the "Read and write to any database" role because this is too permissive (we already have a dbadmin user).
10. To get the connection string, click on **Databases** in the sidebar.

11. Click on **Connect** and select **Connect using MongoDB Compass**.

Export local MongoDB data

1. Log in to MongoDB Compass using your local DB user credentials.
2. Navigate to the collection you want to export (e.g. "menuLinks" from our previous Node app example).
3. Click on the export collection button.
4. In the popup, select "Export Full Collection".
5. Make sure all fields are selected, then click **Select Output**.
6. Select "JSON" as the file format and browse to a location where you want to save the exported JSON file.
7. Click **Export**.

Import JSON to remote MongoDB database

1. Connect to your remote MongoDB database using MongoDB Compass.
    a. From Atlas, copy the connection string to use with MongoDB Compass.
    b. In MongoDB Compass, select **New Connection**.
    c. Click on **Edit** so that you can paste the connection string in the text field.
    d. Paste in the connection string but modify the values accordingly. You need to edit the db user, password and change the database to "testdb". You can get rid of extra options ("?..." stuff). It should look something like the following (replace <pwd> with your remote testdbuser's password and <cluster_host> with cluster address):

    ```
    mongodb+srv://testdbuser:<pwd>@<cluster_host>/testdb
    ```

    > *Primary cluster*
    >
    > If you get an error when trying to import (i.e. a "not primary" error), make sure you're using the primary cluster in the connection string. To find the primary cluster, click on **Databases** in the sidebar then click on the cluster name (e.g. Cluster0). This will open the overview page.
    >
    > Copy the primary cluster name and use that in the connection string as the host name.

    e. Click **Connect** and you should be connected to the remote cluster.
2. Navigate to the menuLinks collection.
3. It should be empty. Click on **Insert Document**.
4. In the popup, click on the button with curly brackets so you can just enter JSON.
5. Delete the current data in the window.

6. Open your exported ***menuLinks.json*** file in an editor and copy/paste the JSON data into the window.

7. Click on the **Insert** button at the bottom of the window (it should be green now).

Modify project

Modify your project files to use the new connection string and user credentials. Use the dotenv module to keep your secret user values from being exposed in the code (see page 4 of the *Secure Local MongoDB* PDF).

Task runners and bundlers

To save time during development, you can use task runners (e.g. Gulp or Grunt) to help with repetitive tasks (for example, minifying CSS or compiling Sass files). You can also use bundlers to combine JS code into one file (e.g. Webpack). Think about how many file loads can be saved if code can be bundled.

To learn more: https://blog.logrocket.com/node-js-task-runners-vs-module-bundlers/