Deploying a Node.js + PostgreSQL app to Heroku

Aemiro Allison Follow
Oct 5, 2017 · 8 min read

Heroku and Node.js Logos

Heroku allows developers to deploy their local apps onto their hosting platform. They have made getting Node.js apps up and running relatively easy. In this guide, I will walk you through deploying your Node.js app on Heroku with a PostgreSQL database.

In this guide, we will be using the following:

- **Node.js:** *javascript server run-time*
- **npm:** *Node package manager*
- **dotenv:** *npm package for loading environment variables into Node*
- **git:** *version control*

- **Heroku:** *free hosting platform*
- **Heroku CLI:** *command line tool to run Heroku commands*
- **PostgreSQL:** *Heroku offers free a postgresql database add-on*

> This guide assumes that you have been working on your app in a git repository , have node and npm installed, and a postgresql database running locally. All commands in this guide were used on a Mac laptop. For Windows users, install the Linux Subsystem for Windows feature in the Windows Anniversary or Creators update, to add support for unix commands on Windows.

. . .

Configuring App Locally

To begin, in your `package.json` file, which should be in the root directory of your project, Heroku requires you to define the ***engines*** section. The ***engines*** section inside `package.json` should include the version of node, *and optionally the version of npm*, that your app is currently using. Run `node -v` and `npm -v` to get your current versions of node and npm. Heroku uses this engines section to determine which versions of node and npm it should run your app on:

```
{
  "name": "your_app",
  "version": "1.0.0",
  "description": "your description...",
  "main": "server.js",
  ...
  "engines": {
    "node": "8.5.0",
    "npm": "5.3.0"
  },
  ...
}
```

. . .

Using Environment Variables

If you have any sensitive data in your app, such as API keys or secrets, defining them in your app is dangerous. If anyone were to gain access to your server, or your source code, they would be able to view all of your sensitive data. For that reason, you will need to create a `.env` file which should contain all your sensitive data, stored as key value pairs for example:

```
// .env file

SECRET=2qe2e21e31

API_KEY=nam$dpoaisjpo
```

Note: If your using React on the front-end, your experience using environment variables inside of your React app may vary. You have a couple options that we won't go too deep into:

- *If your using create-react-app , create a `.env` file in the React project root and inside, all your environment variables must begin with `REACT_APP_` . So if you had a env variable called `SECRET` , it would be written as `REACT_APP_SECRET` etc. Also, remember to restart your node script for it to take affect. More details here: https://medium.com/@tacomanator/environments-with-create-react-app-7b645312c09d*
- *Or, you can use webpack npm package and setup the environment variables in your webpack config file.*

Accessing Environment Variables with 'process.env'

To access these environment variables you declared inside your app, you should use the process.env object. The **process.env** object is a global object in Node that contains all the environment variables that have been set locally. To access our environment variables using `process.env` , we have to load them into in ourselves. Wait, what?! Didn't we just set them up? Well… we just made the `.env` file that contains our environment variables, but we haven't done anything to them yet. So now, to gain access to them on the `process.env` *object,* they need to be loaded. The npm package, dotenv, will take care of loading our environment variables into our application. Install the `dotenv` package with:

```
npm install dotenv --save
```

The `dotenv` package requires you to call it at the beginning of the execution of your app. This will load the environment variables inside of your `.env` file into the `process.env` object so you can use them like `process.env.SECRET`. The dotenv package maintainers recommend executing the package at the top of the first file to run inside your app; probably a *`server.js`*, *`app.js`* or something of that sorts. To configure the package, run the following snippet at the top of your selected file.

```
// load all env variables from .env file into process.env object.
require(‘dotenv’).config()
```

After following the steps above, you should have finished setting up your app’s package.json and configured your app to use environment variables inside of your .env file.

. . .

Setting up Heroku

We’ve finally reached the heroku configuration part of this guide. To being setting up Heroku, you’ll need to first sign up for an account on Heroku. After you’ve completed the sign up process, you’ll then need to install the Heroku CLI to continue. Install it using:

```
npm install -g heroku-cli
```

To test if it has been installed, do `heroku version` *and it should show the current version of the heroku-cli if installed.*

. . .

Logging into Heroku CLI

To use the heroku CLI, you will be prompted every time you run a command, for your email and password. To prevent this from happening every time, you can just login once using `heroku login` and it will prompt you to enter your credentials.

. . .

Linking your app with Heroku CLI

After logging into heroku CLI, you can now tell heroku to create a server for your app that you want to deploy. To create a new server for your app, **inside of the root directory of your app**, run the following:

```
heroku create <app_name>
```

This does a couple things. It gives you a server on heroku, generates a url in the format of: ***https://<app_name>.herokuapp.com,*** for your app that's going on Heroku and adds a remote to your local git repo called `heroku` . You can check if this new remote `heroku` was created by running `git remote -v` .This new git remote `heroku` allows you to use git and push your app onto your server using `git push heroku master` , and if your using a different branch, use `git push heroku` **`branch_name`**`:master` .

Ok, so to check that your server has been created and running, first, save all your current changes by using git to `add` and `commit` your changes, then using the command mentioned before, push your app to the server using `git push heroku master` . After Heroku has finished building your app on the server, use `heroku ps:scale web=1` , which will start up your server if it was not running before.

Quick Note:

The `.env` *file that you have created should NOT be stored on your newly created server for security reasons. Instead,* ***we will need to set each of our environment variables for our app using*** `heroku config:set` **`name=value`**. *So for example, we can set a secret key environment variable using:*

```
heroku config:set SECRET=some_random_string
```

and heroku will take care of loading your environment variables into the `process.env` *object.*

. . .

Let's Recap

So far, we've:

- taken care of using environment variables inside our app, both on heroku and locally.
- added the **engines** section to your `package.json`
- initially set up our heroku server

Now, we're going to focus on getting our app to actually work on our server, and also adding a Postgres database.

. . .

Configuring our App on the server

So using `git push heroku master` or `git push heroku <branch_name>:master` , we can push our app onto heroku, but your app will not work as is. We need to tell heroku to build our app every time we push a new change to the server. We can do this by adding a **heroku-postbuild** script in the **scripts** section of your app's `package.json` . Heroku provides this hook, which runs immediately after Heroku has finished pushing your app onto the server. I've included some common examples below for specific types of Node apps:

For Node + React App:

*Note: You should rename your react app to '**client**' first. It's best practice to name your react app folder 'client' because react is used on the client.*

```
"heroku-postbuild": "npm install && cd client && npm install --only-
dev && npm install && npm run build"
```

For Node only App:

```
“heroku-postbuild”: “npm install --production”
```

These scripts should take care of getting your app ready each time you push a new change to your server. So, we've come a far way with configuring our server. Now, the only thing left do is to add the postgresql database to our app.

. . .

Provisioning the PostgreSQL Database

There are two ways we can add a Postgres database add-on to our app. The easiest way is to use the heroku CLI and add it using:

```
heroku addons:create heroku-postgresql:hobby-dev
```

This tells heroku to add a Postgres database to your app and use the **hobby-dev** plan (a free plan that gives limited database storage). You can check using `heroku addons` , to see if your Postgres database was added.

The other method of adding a Postgres database add-on is to use Heroku's website. Log into your heroku account and go to Heroku addons page. Click on the "**Heroku Postgres**" add-on, then click on the "Install Heroku Postgres". Then it should prompt you to select one of your apps to add the database to. Choose your app and it should ask you what plan you would like to use with this add-on. If you want the free plan, choose the "**Hobby-Dev**" plan and finally click on "**Provision**", which should install the Postgres database. Once your Postgres database add-on has been installed, use `heroku addons` , to check whether or not it's been installed.

After we've set up our database, the only thing left now is to create our tables and add data to our database if needed.

. . .

Creating our tables (Running migration/seed files)

If you have any migration or seed files, Heroku allows you to run them on your Postgres database. To do so, first change directory into the directory that holds the file you want to run eg. `cd db/migrations/` . Then run the file on your database using:

IMPORTANT: *Remove any connects,* `\c <database_name>` *, in the file/s that your trying to run against the database. Heroku creates a database with a random name when you add the postgresql database add-on. Therefore, having any connects will cause an error to occur because your connects will be pointing to your local database's name.*

```
cat <file_name> | heroku pg:psql
```

Let me explain for a second what this code snippet is doing. At the beginning, `cat <file_name>` is getting the contents from the file you want to run against the database and the `|` *pipe* feeds the contents of that file as an argument to the next command, `heroku pg:psql` . This command tells Postgres to use the psql console to run everything that was inside of the file you included in the `cat <file_name>` statement.

So, if you have any other files you want to run on your database, you can follow the same process:

1. First, Change directory into the directory of the file you want to run.
2. Then, run: `cat <file_name> | heroku pg:psql` to apply changes to your database.

. . .

Finally viewing your Heroku app

- To see your newly created app, do `heroku open` which should open up your app in your default browser.
- To run your heroku app locally, use `heroku local` and it should start up a node server locally at `http://localhost:5000` only if you have not set a port for your server. **If you have set a port for your server**, go to `http://localhost:<your_port_here>` to test your app.

. . .

Conclusion

Give yourself a pat on the back! You should now have a Node.js app with a postgresql database running on Heroku. This article mainly used the official documentation provided by Heroku on setting up a Node.js app and adding a Postgres database.

If you have any feedback or criticism, leave a comment. It's my first tutorial/guide so bear with me 😃.

Heroku Node React Postgres Deployment