You are welcome to review this document, but it has been replaced by these instructions:

https://docs.google.com/document/d/1PwGR-QbVKhABIwpYro97DFe0KQkfG8MqP_K6mHjdYLI/edit?usp=sharing

CSSE 280: Deploying your web app to Heroku and mLab

Objectives

You will be able to

- Deploy your server API application to Heroku - https://www.heroku.com
- Deploy your database to mLab - https://mlab.com/
- Connect your server back-end application to your mLab hosted database

Table of Contents

Introduction

A concern for Node.js applications developers is deploying their applications to a production environment. Heroku is one of the cloud platform as a service (PaaS) providers that helps solve this problem. Heroku allows you to deploy your Node.js application onto the real Internet for free. No more pointing your browser to http://localhost:3000.

How to submit your work

This will be graded as a homework assignment. Be sure to turn in the ***herokuapp.com*** URL for your back-end API server application in the Web App Deployment Submission spreadsheet.

Getting our database live on mLab

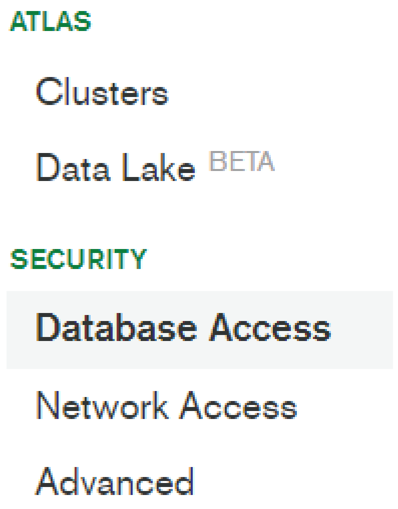
If your application is live on the Internet, it is pointless having the database on your local machine. Your database also needs to be externally accessible. In this section you're going to push your database into a live environment and update your REST API to use the published database from the published site.

Setting up mLab manually

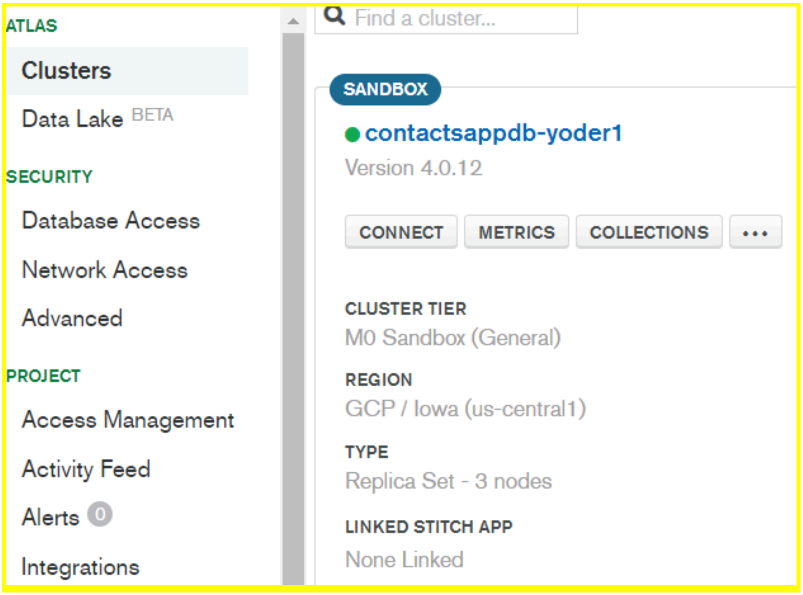
This is the preferred approach since using an add-on from Heroku approach would require you to submit your credit card.

http://docs.mlab.com/ has provided a Quick-Start guide to mLab, but the most important steps are highlighted below.

1. Sign up for a free account from https://mlab.com/
2. Click on the **Create new** button under MongoDB Deployments to create a new database
3. Choose **Google Cloud Platform** or **Microsoft Azure** as your Cloud Provider
    a. **Do not use Amazon web services**, use a different one (Google or Azure)
4. Choose **Sandbox**, which is free, as your Plan Type
    a. Click the Continue button at the bottom of the page.
5. Choose a **Region** and click on the **Continue** button at the bottom of the page
6. Enter database name ***contactsappdb-<Rose-username>*** in the appropriate text field then click **Continue**
7. Verify your **Order Confirmation** then click the **Submit Order** button at the bottom of the page.
8. Click on the ***Home*** link (top left), then click on the name of the database below the ***MongoDB Deployments*** heading and ***Deployment Utility*** subheading.
9. Add at least one user by clicking the ***Users*** tab
    a. Click the ***Add database user*** button.
    b. Enter the new user credentials, then click ***Create***.



Add to Whitelist



contactsappdb-yoder1
Version 4.0.12

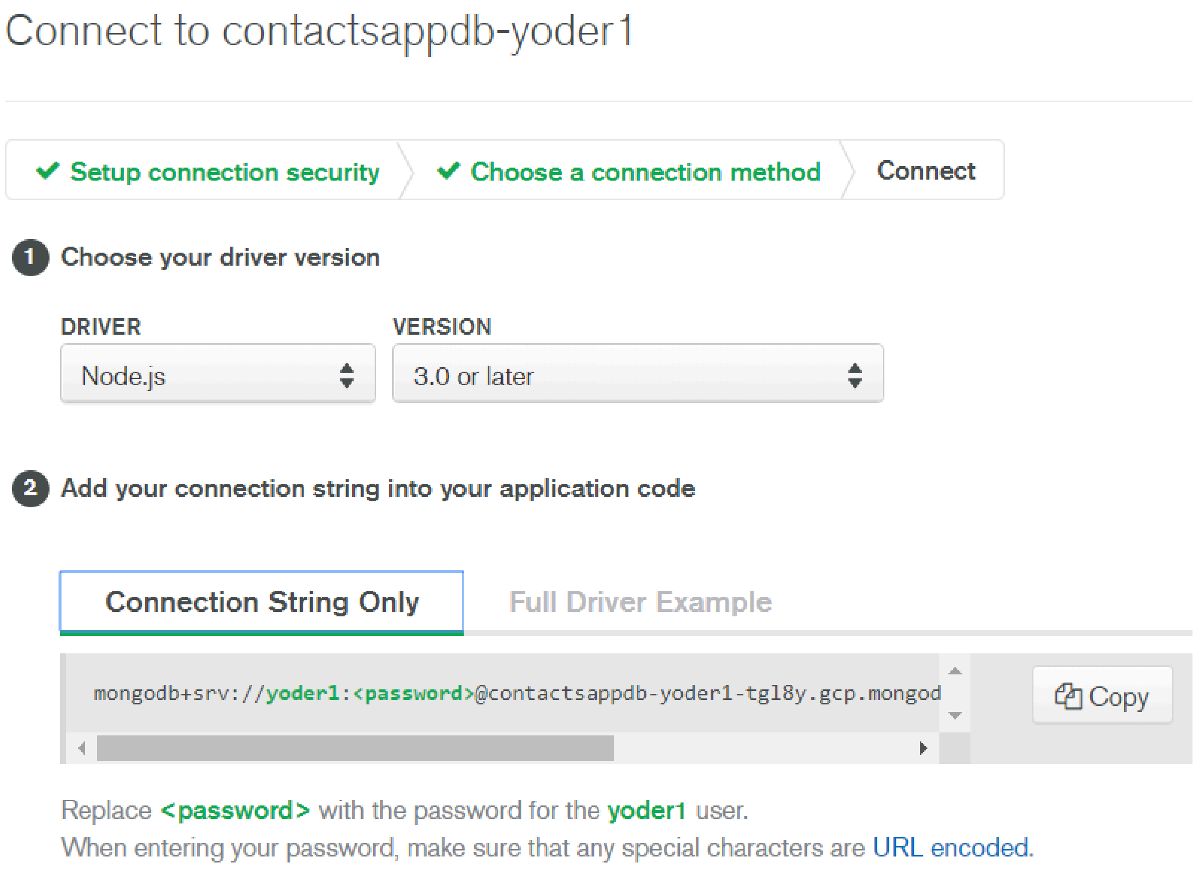
CLUSTER TIER
M0 Sandbox (General)

REGION
GCP / Iowa (us-central1)

TYPE
Replica Set - 3 nodes

LINKED STITCH APP
None Linked

10. **Get the database URI (connection string). This step is very important.**
    a. Click the database name to get the database URI.
    b. It should look something like ***mongodb://dbuser:dbpassword@dsyour_port_number.mlab.com:your_port_number/contactsappdb-<Rose-username>***
    c. **Most of the parts will be different for you and you will have to swap out the username and password with what you specified in step 9b.**
    d. To connect using the mongo shell enter the following at a terminal



Connect to contactsappdb-yoder1

1 Choose your driver version

DRIVER: Node.js

VERSION: 3.0 or later

2 Add your connection string into your application code

```
mongodb+srv://yoder1:<password>@contactsappdb-yoder1-tgl8y.gcp.mongod
```

Replace <password> with the password for the yoder1 user.
When entering your password, make sure that any special characters are URL encoded.

> ***mongo dsyour_port_number.mlab.com:your_port_number/conmongtactsappdb-<Rose-username> -u dbuser -p dbpassword***

Again, swap out the username and password with what you specified in step 9b.

You should see a confirmation that you are connected to the database. E.g.,

MongoDB shell version: 3.6.8
connecting to: ds031257.mlab.com:31257/contactsappdb-<Rose-username>

--Note: the above may not work with git bash if you had problems with your mongodb installation. In this case you may want to do this in PowerShell.

You should now be able to interact (on a limited basis) with the database via the mongo shell. You might not be able to do much in the shell for security reasons. You might want to explore the website instead.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Migrating your local database to mLab

Now that you have set up a remote database and know all the details for connecting to it, you can migrate your local database to it by following the steps below.

1. Create a temp directory to store your **local database data dump**
2. Dump the data from your local database in that directory
3. Restore the data to the remote database
4. Test the remote database

These steps can be executed from a bash-enabled terminal window (e.g., GitBash on Windows).

Creating a temp directory

```
> mkdir -p tmp/mongodump
```

Dump the data from your local database

You should make sure that ***mongod*** is running in a different terminal when you execute this command.

```
> mongodump -h localhost:27017 -d contactsappdb -o tmp/mongodump
```

-h - The hostname and port number
-d - the local database name
-o - the destination directory for the data dump

https://docs.mongodb.com/manual/reference/program/mongodump/ lists additional parameters.

Restore data to remote database

```
> mongorestore -h ds031257.mlab.com:31257 -d contactsappdb-<Rose-username> -u
dbuser -p pword tmp/mongodump/contactsappdb
```

-h - remote hostname and port number
-d - remote database name (how you want to name the db on mlab.com)
-u - username for remote database
-p - password for remote database
path to the dump directory and database name

These parameters will all be different for you. Replace them with the values from your connection string.

https://docs.mongodb.com/manual/reference/program/mongorestore/ => additional parameters.

Testing the remote database

```
> mongo ds031257.mlab.com:31257/contactsappdb-<Rose-username> -u dbuser -p dbpassword
```

Again, **these values will be different for you**. Don't forget to swap out the username and password with what you specified above.

You should see confirmation in the terminal window that looks like this:

```
MongoDB shell version: 3.6.8
connecting to: ds031257.mlab.com:31257/contactsappdb-<Rose-username>
```

You can run a few ***simple find() queries*** on the remote database to verify that data did get transferred.

Getting Heroku setup

Before you can use Heroku, you need to sign up for a free account at https://www.heroku.com/. Choose ***Node.js*** as your **Primary Development Language**.

Next, you'll want to install the Heroku Command Line Interface (CLI) (used to be called the Heroku Toolbelt) by following installation instructions for your platform from https://devcenter.heroku.com/articles/heroku-command-line. This will install

1. ***Heroku client:*** a CLI for managing Heroku apps. You'll use it to manage your Express applications.
2. ***Forego:*** Another CLI that you'll use to define how you want your applications to run.
3. ***Git:*** a version control system that you already have installed.

Note: For windows users, will need to add the bin directory to your system environment path. System > Advanced System Settings > Environment Variables > Path > Edit > New , Then copy your file location to heroku's bin directory (default is C:\Program Files\Heroku\bin).

Once you have everything set up, we can continue and get your app ready to go live.

Making a Heroku ready app

Heroku can run applications on all different types of codebases, so we need to tell Heroku what our application is running. We need to tell it that we are running a ***Node.js application*** and are using ***npm*** as the Node.js package manager. The version numbers for these are very important since we want to ensure that the production setup is the same as the development setup. To get the version numbers for these, run the following commands in a terminal. We will use > to indicate the prompt.

TO MINIMIZE ISSUES ON WINDOWS SYSTEMS, ALL REMAINING TERMINAL COMMANDS SHOULD BE RUN IN POWERSHELL.

```
> node -v
> npm -v
```

From the terminal, change directory (***cd***) to the ***root directory of your completed back-end API server application (contacts-app-backend-with-middleware)***.

Adding an engines section to manifest

Open the ***package.json*** file in VS Code or your favorite editor and add the following section.

```
"engines": {
        "node": "~8.11.0",
        "npm": "5.5.0"
},
```

--Note, you may need to retype the “” as copy/paste can be weird.

When pushed up to Heroku, this will tell Heroku that our application uses the latest patch version of Node.js, 8.11, and the latest patch version of npm, 5.6.

See https://devcenter.heroku.com/articles/nodejs-support for the versions of Node.js and npm that Heroku supports. Make sure you are specifying from among these versions.

We should start our application on ***const port = process.env.PORT* || 3000 (in app.js)** because Heroku stores a port number for the application in this environment variable. Note: You can use 3000 or any port number that you wish. The point to note here is that Heroku may opt to provide you a port number that is different from the one you choose and will default to using that instead.

Change your port number in app.js in your backend to:

```
const port = process.env.PORT || 3000
```

Creating a Procfile

The package.json manifest tells Heroku that the application is a Node.js application, but it does **not** tell Heroku how to start it. To tell Heroku how to start our application, we need to define a **Procfile**. This will declare the process types of our application and the commands used to start them.

In the backend directory of the application, create a file called ***Procfile*** (this is case sensitive and has no extension). Type the following line in the ***Procfile***.

web: node app.js

OR

web: node ./bin/www // if this is where your app starts

When pushed to Heroku, this file tells Heroku that the application needs a ***web process*** and that it should ***run node app.js OR node ./bin/www***.

Testing app locally with heroku local (May not work on Windows)

We can use heroku local to verify our setup and run our application locally before pushing the application to Heroku. **This part of the lab may not work on WIndows**.

If the application is currently running, stop it by pressing ***Ctrl-C*** in the terminal window that's running the process. Now, in the terminal window, enter the following command.

> ***heroku local***

If all the above steps went well, this will run the application locally on possibly a different port (might be 5000 instead of 3000). You should see confirmation of that in the terminal. Now open your browser and enter the address http://localhost:5000 (or the confirmed port number) in the address bar to verify that the app still works.

Note: If nothing happens or it errors out, it's not a huge problem; move to the next step.

Pushing the site live using Git

Heroku uses Git as the deployment method. If you need a quick review, Git Tutorial - Try Git at https://try.github.io is an excellent reference.

Storing the app in Git

The first action toward pushing your site live using Git is to store the application in Git, on your local machine. This can be done as follows:

1. Add a **.gitignore** file that tells git to ignore certain files and directories. Create a file in the application directory (e,g., **contacts-app-backend-with-middleware**) called ***.gitignore*** (Note the dot at the start of the file name and the fact that the file has no extension) and add the following content to the file.
   node_modules

2. Initialize the **application folder** as a Git repository by using the following terminal command:
   > ***git init***

3. Tell Git which files to add to the repository by using the following command:
   > ***git add .***

4. Commit these changes to the repository using the following command:
   > ***git commit -m "Initial commit"***

These commands together will create a local repository containing the entire codebase for the application.

Logging in and Creating the Heroku application

This step will create an application on Heroku, as a remote Git repository of your local repository. From the terminal, enter the following commands, in order, responding to prompts as needed. This will log you into Heroku from the CLI and create the Heroku application. In the commands below, replace ***username*** with your ***Rose-Hulman network username*** or use a totally different ***unique*** name for your app.

> ***heroku login***
> ***heroku create contacts-app-api-username***

If you do not include a name for your app, Heroku will assign your app some arbitrary name. You should see a confirmation of that in the terminal. Now login to your Heroku account in a browser to see that the application exists. Click on the app in the browser, then on the Settings tab to see the Heroku Git URL for your app; copy it. Now add the git remote so you can work toward posting your app to Heroku.

> ***git remote add heroku PASTE_THE _HEROKU_GIT_URL_HERE***

If you get an error from this command saying "remote heroku already exists" it is not an

error. It just means that the remote was already added.

```
> git remote -v
```

You should see the your heroku remote with url(s) listed.

Deploying the application to Heroku

Now, tell your newly created Heroku app that it is a production Node.js environment.You do so by setting the NODE_ENV environment variable on the Heroku servers. This can be accomplished by entering the following command in the terminal.

```
> heroku config:set NODE_ENV=production
```

Your application is still stored in the local Git repository and the remote repository you've created on Heroku is empty. So you need to push the content of your local repository to the ***heroku*** remote repository by entering the following command in the terminal.

```
> git push heroku master
```

This will log a lot of messages to the console as it goes through the process, eventually ending up with a confirmation that the application has been deployed to Heroku. E.g.,

```
https://contacts-app-api-username.herokuapp.com deployed on Heroku
```

The next step is to tell Heroku to run your app with one process, so it will actually run on a real computer. This can be accomplished by running the following in the terminal.

```
> heroku ps:scale web=1
```

A confirmation will display in the console and your application will be live on the Internet.

By the way, you can run the following command to see your environment variables on heroku:

```
> heroku config
```

To view your application on a live URL, enter the following command in the terminal.

```
> heroku open
```

This should launch the application in your default web browser. But wait! There may be a problem. It needs to access your database, which is only accessible from your local machine. The section Getting our database live on mLab addresses this problem.

Updating and redeploying your Heroku application

Now that your Heroku application is setup, updating and redeploying it is easy. Every time you want to push some new changes through, you need to enter the following terminal commands from the root directory of the application.

```
> git status
> git add .
> git commit -m “commit message goes here”
> git push heroku master
```

Updating Heroku configuration with mLab Database connection string

Once you have your full connection string for your mLab database, you should save it as part of your Heroku configuration. In the ***root directory of your API application***, enter the following commands in a terminal window.

```
> heroku login  // make sure you are still logged in
> heroku config:set MLAB_URI=your_db_uri
```

Replace ***your_db_uri*** with your full connection string, including the ***mongodb://*** protocol.

The last command should look something like this:

```
> heroku config:set MLAB_URI=mongodb://dbuser:dbpassword@dsyour_
port_number.mlab.com:your_port_number/contactsappdb-<Rose-username>
```

Replace the appropriate parts of the URI as needed. The best way to get the correct URI is to login to https://mlab.com, click on your specific MongoDB Deployment (database), copy and paste your MongoDB URI, replacing dbuser and dbpassword with your mLab credentials.

Recall that you run the following command to see your heroku environment variables:

```
> heroku config
```

Setting the database URI based on the environment

The database connection for the back-end API application is held in ***models/db.js***. The connection portion of the this file currently looks like the code snippet below.

```
const dbURI = `mongodb://${server}/${database}`;

...

mongoose.connect(dbURI, options).then().catch(); // with options values
```

We need to update this to use the connection string set in the heroku environment variables. The code snippet for the first line above should be replaced as follows:

```
let dbURI = `mongodb://${server}/${database}`;
if (process.env.NODE_ENV === 'production') {
   dbURI = process.env.MLAB_URI;
}
```

Testing before launching (May not work on Windows)

In the ***root directory of your back-end API application***, enter the following command in a terminal window to test the changes locally before launching on Heroku.

Note: This may **not** work on **Windows** because Windows does not allow you to declare an environment variable from the terminal this way. If you get an error, it is okay to skip this step.

```
> MLAB_URI=your_db_uri
> NODE_ENV=production
```

Don't forget to replace ***your_db_uri*** with your actual MongoDB URI on mLab.

You should see a confirmation that looks something like below:

```
> node app.js

listening on port 3000
Mongoose connected to your_db_uri
```

Launching on Heroku

If your local tests are successful and you can connect to your remote database by temporarily running your application in production mode, then you are ready to push your changes to Heroku. Follow instructions from Updating and redeploying your Heroku application to do so. Don't forget to run

> ***git push heroku master***.

You can run the following command in the terminal to look at the Heroku logs and verify that your application is connected to your remote database. If you see errors, address them and update/redeploy your Heroku application.

```
> heroku logs
```

Your Heroku logs are also available from the Heroku's website. When you login and access your app, click on the **More** dropdown list and select the View Logs option.

Challenge Exercise (optional)

Now that you know how to deploy a frontend application to Firebase, a database to mLab, and a backend server application to Heroku, do the following with your contacts-app-front application:

- Make it use your Heroku deployed API instead of the localhost version of the API
- Deploy it to Firebase -- review these instructions
- Enter the URL of the frontend in the Web App Deployment Submission spreadsheet

Troubleshooting

Problem: Application Error

My app is not loading in the browser, is taking forever to load, or reporting Application Error.

Resolution to Application Error

Here are some of the things you can do to address this.

```
> heroku login  // login to heroku
> heroku config // will spit out the values of NODE_ENV (and MLAB_URI for backend)
```

If any variable (NODE_ENV or MLAB_URI) is incorrect, set it to its expected values.

```
> heroku logs // gives a console log of the activities of the application and reports errors
```

If you see errors in the log, address them then update/redeploy your Heroku application. One possible error is that ***mongoose*** or another package is missing. To install mongoose and update your package.json manifest run the following command:

```
> npm install mongoose --save
```

Problem setting MLAB_URI

I see errors in the console when setting the MLAB_URI variable.

Resolution to setting MLAB_URI

Make sure there is no **!** character in the username or password. If you included it in your dbuser credentials, correct that from https://mlab.com.

Problem with pushing app to heroku

I have trouble pushing my app to heroku master

Resolution to pushing app to heroku

Here are a few things you can do to address this.

> ***git status*** // status of your repository

If there are files that are not committed, update/redeploy your Heroku application. Do all but the last step (git push heroku master).
> ***git remote -v*** // shows remote repositories that the app is connected with

If you don't see a confirmation similar to the following,

```
heroku https://git.heroku.com/your_app_name.git (fetch)
heroku https://git.heroku.com/your_app_name.git (push)
```

find the name of your app by login into https://www.heroku.com. Copy the name of your app (not the full URL) and run the following command at the terminal.

> ***git remote add heroku https://git.heroku.com/your_app_name.git*** // add missing remote
> ***git remote -v*** // confirm remote is added

If you get an error because the remote is already added, this is because you only need to add the remote once. If you wish to change the remote, here is how you do so.

> ***git remote set-url heroku https://git.heroku.com/your_correct_app_name.git***
> ***git remote -v*** // confirm remote is added

> ***git push heroku master*** // now push your app to heroku and heroku will deploy it for you.

Problem with Windows console

Git-bash or Command Prompt is giving me problem interacting with my app on Windows.

Resolution to Windows console

Consider installing the ConEmu (https://sourceforge.net/projects/conemu/) or cmder terminal emulator (http://cmder.net/). You might want to download and install the ***full*** as opposed to the mini version of the application. Use this as your terminal instead.