Class 13: Deployment

The focus of this class is on students understanding:

- the concept of production deployments
    - in conjunction with development and production environments.

Through the use of Heroku, you will guide students through:

- the deployment process,
- configuration of environment variables,
- and general issues that can arise during the deployment phase of a project.

Students should also understand development, production, and feature branches and how they are used in collaborative development.

Class 13 Learning Objectives

- Be able to push a dev site to production, so the world can see it.
- Understand the difference between a static page and a dynamically generated app page.

Topic 1 - Environments

- Dev vs Production environment
- What we look for in production env
- What to do with the GitHub token?
- How to deploy to Heroku (step by step)

Demo

- Create free account
    - https://signup.heroku.com/dc
- Set up
    - In this step you will install the Heroku Command Line Interface (CLI).
    - You will use the CLI to:
        - manage and scale your applications,
        - to provision add-ons,
        - to view the logs of your application as it runs on Heroku,
        - as well as to help run your application locally.
    - Install CLI

- Link
- Once installed, you can use the heroku command from your command shell.
- Log in using the email address and password you used when creating your Heroku account:

```
    heroku login
    Enter your Heroku credentials.
    Email: zeke@example.com
    Password:
    ...
```

Authenticating is required to allow both the heroku and git commands to operate.

Before you continue, check that you have the prerequisites installed properly. Type each command below and make sure it displays the version you have installed. (Your versions might be different from the example.) If no version is returned, go back to the introduction of this tutorial and install the prerequisites. All of the following local setup will be required to complete the “Declare app dependencies” and subsequent steps. This tutorial will work for any version of Node greater than 4 or so - check that it’s there:

```
$ node -v
v5.9.1
```

npm is installed with Node, so check that it’s there. If you don’t have it, install a more recent version of Node:

```
$ npm -v
3.7.3
```

Now check that you have git installed. If not, install it and test again.

```
$ git --version
git version 2.2.1
```

Prepare the app

In this step, you will prepare a simple application that can be deployed. To clone the sample application so that you have a local version of the code that you can then deploy to Heroku, execute the following commands in your local command shell or terminal:

```
$ git clone https://github.com/heroku/node-js-getting-started.git
```

```
$ cd node-js-getting-started
```

You now have a functioning git repository that contains a simple application as well as a package.json file, which is used by Node’s dependency manager.

Deploy the app

In this step you will deploy the app to Heroku. Create an app on Heroku, which prepares Heroku to receive your source code.

```
$ heroku create
Creating sharp-rain-871... done, stack is cedar-14
http://sharp-rain-871.herokuapp.com/ | https://git.heroku.com/sharp-rain-871.git
Git remote heroku added
```

When you create an app, a git remote (called heroku) is also created and associated with your local git repository.

Heroku generates a random name (in this case sharp-rain-871) for your app, or you can pass a parameter to specify your own app name.

Now deploy your code:

```
$ git push heroku master
Counting objects: 343, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (224/224), done.
Writing objects: 100% (250/250), 238.01 KiB, done.
Total 250 (delta 63), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Node.js app detected
remote:
remote: -----> Creating runtime environment
remote:
remote:        NPM_CONFIG_LOGLEVEL=error
remote:        NPM_CONFIG_PRODUCTION=true
remote:        NODE_MODULES_CACHE=true
remote:
remote: -----> Installing binaries
remote:        engines.node (package.json):  5.9.1
remote:        engines.npm (package.json):   unspecified (use default)
remote:
remote:        Downloading and installing node 5.9.1...
remote:        Using default npm version: 2.7.4
   ....
remote: -----> Build succeeded!
remote:        ├── ejs@2.4.1
remote:        └── express@4.13.3
remote:
remote: -----> Discovering process types
remote:        Procfile declares types -> web
remote:
remote: -----> Compressing... done, 9.4MB
remote: -----> Launching... done, v8
remote:        http://sharp-rain-871.herokuapp.com deployed to Heroku
To https://git.heroku.com/nameless-savannah-4829.git
 * [new branch]      master -> master
```

The application is now deployed. Ensure that at least one instance of the app is running:

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

Now visit the app at the URL generated by its app name. As a handy shortcut, you can open the website as follows:

```
$ heroku open
```

View logs

Heroku treats logs as streams of time-ordered events aggregated from the output streams of all your app and Heroku components, providing a single channel for all of the events. View information about your running app using one of the logging commands, heroku logs --tail:

```
$ heroku logs --tail
2011-03-10T10:22:30-08:00 heroku[web.1]: State changed from created to starting
2011-03-10T10:22:32-08:00 heroku[web.1]: Running process with command: `node
index.js`
2011-03-10T10:22:33-08:00 heroku[web.1]: Listening on 18320
2011-03-10T10:22:34-08:00 heroku[web.1]: State changed from starting to up
```

Visit your application in the browser again, and you'll see another log message generated.

Press `Control+C` to stop streaming the logs.

Run the app locally

Now start your application locally using the `heroku local` command, which was installed as part of the Heroku CLI:

```
$ heroku local web
14:39:04 web.1     | started with pid 24384
14:39:04 web.1     | Listening on 5000
```

Just like Heroku, `heroku local` examines the `Procfile` to determine what to run. Open `http://localhost:5000` with your web browser. You should see your app running locally. To stop the app from running locally, in the CLI, press `Ctrl+C` to exit.

Push local changes

In this step you'll learn how to propagate a local change to the application through to Heroku. As an example, you'll modify the application to add an additional dependency and the code to use it. Begin by adding a dependency for cool-ascii-faces in package.json. Run the following command to do this:

```
$ npm install --save --save-exact cool-ascii-faces
cool-ascii-faces@1.3.3 node_modules/cool-ascii-faces
└── stream-spigot@3.0.5 (xtend@4.0.1, readable-stream@1.1.13)
```

Modify index.js so that it requires this module at the start. Also add a new route (/cool) that uses it. Your final code should look like this:

```
var cool = require('cool-ascii-faces');
var express = require('express');
var app = express();

app.set('port', (process.env.PORT || 5000));

app.use(express.static(__dirname + '/public'));

// views is directory for all template files
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');

app.get('/', function(request, response) {
  response.render('pages/index')
});

app.get('/cool', function(request, response) {
  response.send(cool());
});

app.listen(app.get('port'), function() {
  console.log('Node app is running on port', app.get('port'));
});
```

Now test locally:

```
$ npm install
$ heroku local
```

Visiting your application at http://localhost:5000/cool, you should see cute faces displayed on each refresh: (⚆ _ ⚆).

Now deploy. Almost every deploy to Heroku follows this same pattern. First, add the modified files to the local git repository:

```
$ git add .
```

Now commit the changes to the repository:

```
$ git commit -m "Demo"
```

Now deploy, just as you did previously:

```
$ git push heroku master
```

Finally, check that everything is working:

```
$ heroku open cool
```

You should see another face.

Setting up config vars for a deployed application

Use the Heroku CLI's config, config:set, config:get and config:unset to manage your config vars:

```
$ heroku config:set GITHUB_USERNAME=joesmith
Adding config vars and restarting myapp... done, v12
GITHUB_USERNAME: joesmith

$ heroku config
GITHUB_USERNAME: joesmith
OTHER_VAR:    production

$ heroku config:get GITHUB_USERNAME
joesmith

$ heroku config:unset GITHUB_USERNAME
Unsetting GITHUB_USERNAME and restarting myapp... done, v13
```

Heroku manifests these config vars as environment variables to the application. These environment variables are persistent – they will remain in place across deploys and app restarts – so unless you need to change values, you only need to set them once. Whenever you set or remove a config var, your app will be restarted.

Define config vars

Heroku lets you externalise configuration - storing data such as encryption keys or external resource addresses in config vars.

`heroku local` will automatically set up the environment based on the contents of the `.env` file in your local directory. In the top-level directory of your project there is already a `.env` file that has the following contents:

```
TIMES=2
```

If you run the app with heroku local, you'll see two numbers will be generated every time.

To set the config var on Heroku, execute the following:

```
$ heroku config:set TIMES=2
```

View the config vars that are set using heroku config:

```
$ heroku config
== sharp-rain-871 Config Vars
PAPERTRAIL_API_TOKEN: erdKhPeeeehIcdfY7ne
TIMES: 2
```