**CIT322 Distributed System**

**Lab3: Intro to Software-as-a-Service, SaaS**

Your name: Ryan Nuebel \_

**Objectives:**

* To understand the concept of Software-as-a-Service
* To learn how to manage resources with the SaaS platform
* To create a simple web app and publish it into a remote cloud

**Prerequisites**

We will be using Heroku cloud for this lab.

* Create an account in Heroku <https://signup.heroku.com/dc>
  + The account should free and you should never be asked to provide any method of payment, if that is the case, please contact me.
  + Make sure you use your pace email address
  + Enter “Pace University” in Company name field
  + Select “Student” in the Role field dropdown.
* Ensure that you have Node.js and npm installed on your machine
  + This should be the case since you completed lab2, however, if this is not the case, make sure both Node.js & npm are installed and you can reference lab2 on how to.
* The Heroku CLI requires **Git**, the popular version control system. If you don’t already have Git installed, complete the following before proceeding:
  + [Git installation](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git)
  + [First-time Git setup](https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup)

## Set up

In this step you’ll install the Heroku Command Line Interface (CLI). You use the CLI to manage and scale your applications, provision add-ons, view your application logs, and run your application locally.

Download and run the installer for your platform:

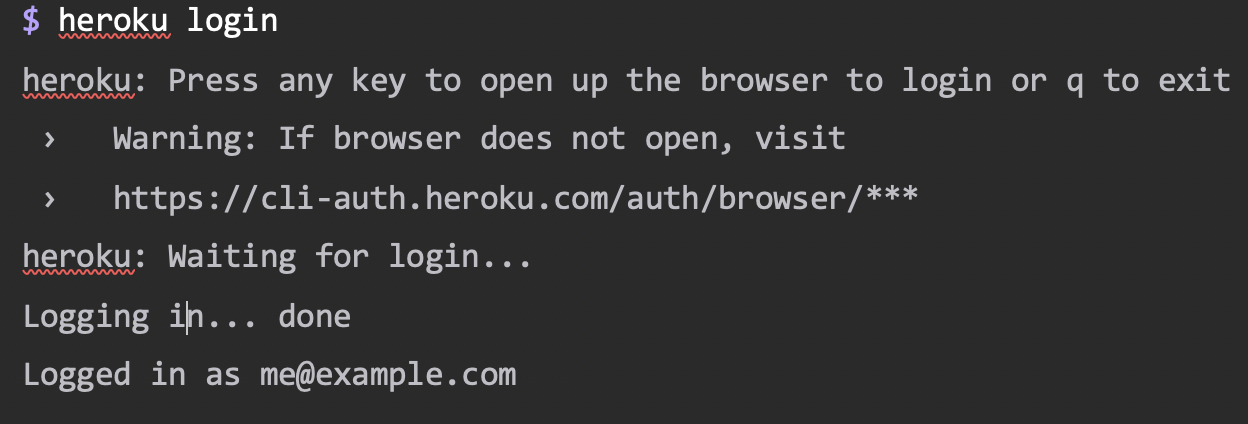
* MacOS: <https://cli-assets.heroku.com/heroku.pkg>
* Windows 64-bit installer: <https://cli-assets.heroku.com/heroku-x64.exe>
* Windows 32-bit installer: <https://cli-assets.heroku.com/heroku-x86.exe>

**Lab Instructions**

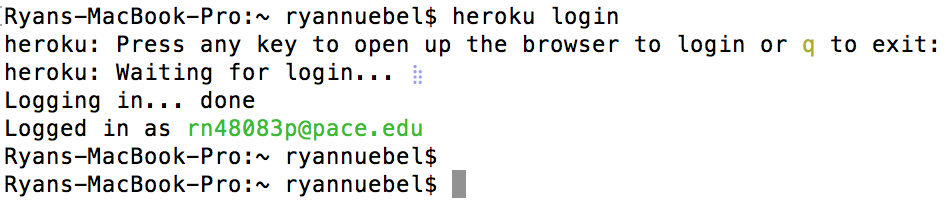
When installation completes, you can use the heroku command from your terminal.

1. **Login to Heroku**

Use the heroku login command to log in to the Heroku CLI:



QA. Paste a screen-shot below of your successful login:



Validate your node, npm, and git version by running the following command and paste screenshot for each one.

$ node --version

v10.13.0

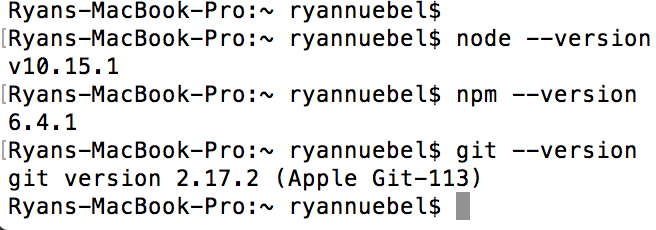
$ npm --version

6.4.1

$ git --version

git version 2.15.1

QB. Paste a screen-shot below of your node, npm & git versions :



1. **Prepare your sample app**
2. **Create your 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

$ heroku create

Creating sharp-rain-871... done, stack is heroku-18

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.

QC. Paste a screen-shot below of your Heroku create command



1. **Deploy your code into Git**

To deploy your app into git, run the following command

$git push heroku master

$ git push heroku master

Counting objects: 488, done.

remote: Verifying deploy... done.

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](https://devcenter.heroku.com/articles/logging), heroku logs --tail:

$ heroku logs --tail

2018-03-10T10:22:30-08:00 heroku[web.1]: State changed from created to starting

2018-03-10T10:22:32-08:00 heroku[web.1]: Running process with command: `node index.js`

2018-03-10T10:22:33-08:00 heroku[web.1]: Listening on 18320

2018-03-10T10:22:34-08:00 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.

**Define a Procfile**

Use a [Procfile](https://devcenter.heroku.com/articles/procfile), a text file in the root directory of your application, to explicitly declare what command should be executed to start your app.

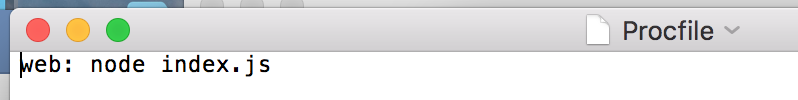
The Procfile in the example app you deployed looks like this:

web: node index.js

This declares a single process type, web, and the command needed to run it. The name web is important here. It declares that this process type will be attached to the [HTTP routing](https://devcenter.heroku.com/articles/http-routing) stack of Heroku, and receive web traffic when deployed.

Procfiles can contain additional process types. For example, you might declare one for a background worker process that processes items off of a queue.

QD. Paste a screen-shot below of your procfile



**Scale the app**

Right now, your app is running on a single web [dyno](https://devcenter.heroku.com/articles/dynos). Think of a dyno as a lightweight container that runs the command specified in the Procfile.

You can check how many dynos are running using the ps command:

$ heroku ps

=== web (Free): `node index.js`

web.1: up 2014/04/25 16:26:38 (~ 1s ago)

By default, your app is deployed on a free dyno. Free dynos will sleep after a half hour of inactivity (if they don’t receive any traffic). This causes a delay of a few seconds for the first request upon waking.

$ heroku ps:scale web=0

Access the app again by hitting refresh on the web tab, or heroku open to open it in a web tab. You will get an error message because you no longer have any web dynos available to serve requests.

Scale it up again:

$ heroku ps:scale web=1

**Declare app dependencies**

Heroku recognizes an app as Node.js by the existence of a package.json file in the root directory. For your own apps, you can create one by running npm init --yes.

The demo app you deployed already has a package.json, and it looks something like this:

{

"name": "node-js-getting-started",

"version": "0.3.0",

...

"engines": {

"node": "8.11.1"

},

"dependencies": {

"ejs": "^2.5.6",

"express": "^4.15.2"

},

...

}

The package.json file determines both the version of Node.js that will be used to run your application on Heroku, as well as the dependencies that should be installed with your application. When an app is deployed, Heroku reads this file and installs the appropriate node version together with the dependencies using the npm install command.

Run this command in your local directory to install the dependencies, preparing your system for running the app locally:

QE. Paste a screen-shot below of your package.json file



$ npm install

added 132 packages in 3.368s

Once dependencies are installed, you will be ready to run your app locally.

**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

[OKAY] Loaded ENV .env File as KEY=VALUE Format

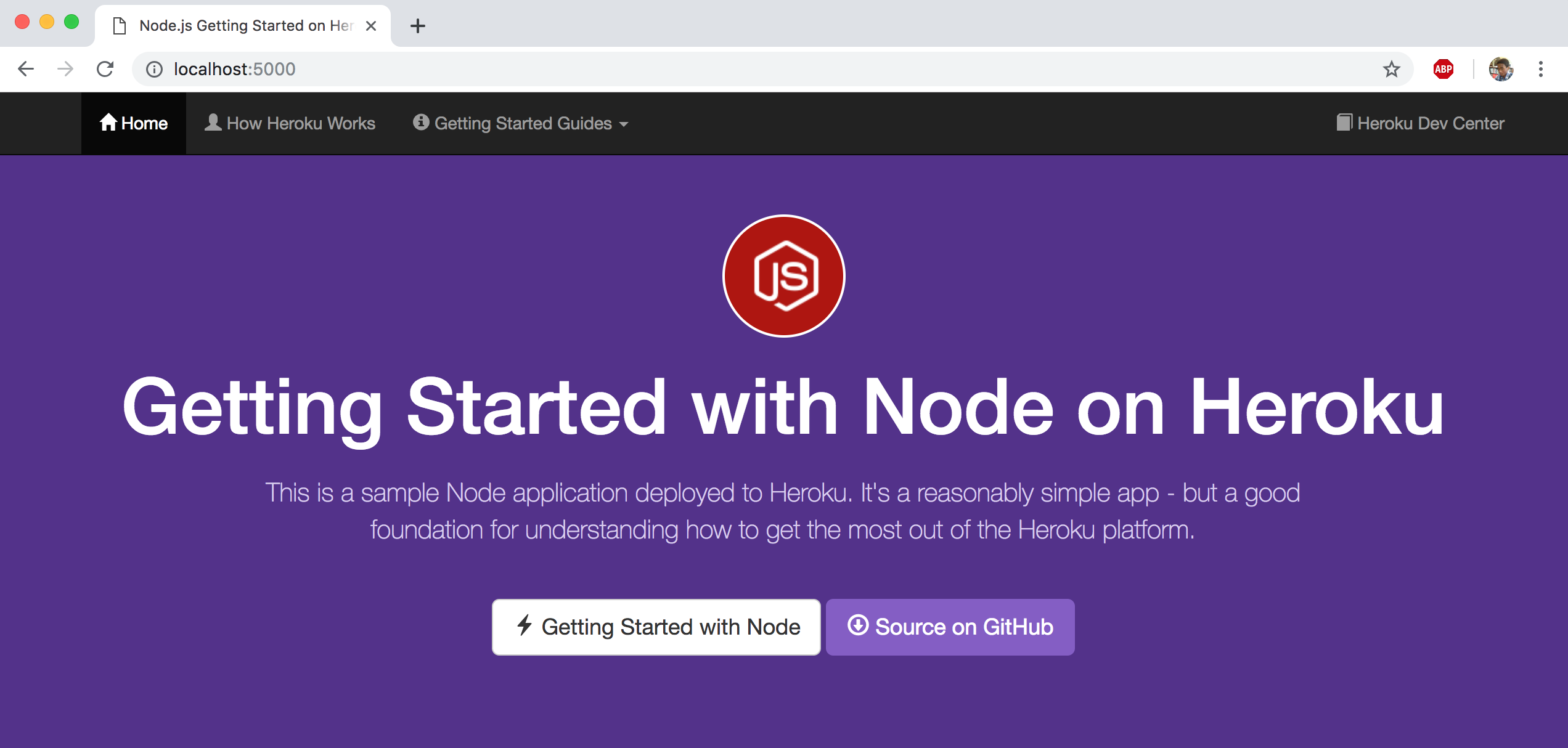
1:23:15 PM web.1 | Node app is running on port 5000

Just like Heroku, heroku local examines the Procfile to determine what to run.

Open [http://localhost:5000](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.

QF. Paste a screen-shot below of your application from your local browser



**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 cool-ascii-faces

+ cool-ascii-faces@1.3.4

added 9 packages in 2.027s

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:

const cool = require('cool-ascii-faces')

const express = require('express')

const path = require('path')

const PORT = process.env.PORT || 5000

express()

.use(express.static(path.join(\_\_dirname, 'public')))

.set('views', path.join(\_\_dirname, 'views'))

.set('view engine', 'ejs')

.get('/', (req, res) => res.render('pages/index'))

.get('/cool', (req, res) => res.send(cool()))

.listen(PORT, () => console.log(`Listening on ${ PORT }`))

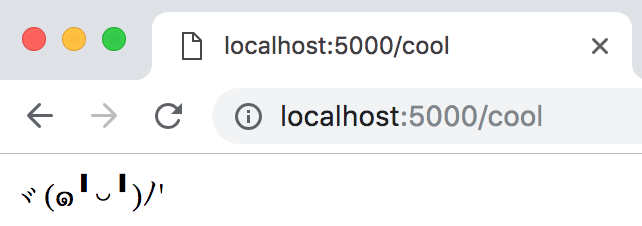
Now test locally:

$ npm install

$ heroku local

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

QG. Paste a screen-shot below of your updated application from your browser



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 "Add cool face API"

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.

QH. What is the URL to access your application?

<it should be in the form of https://<username>.herokuapp.com >

Paste a screen-shot below of your application on Heroku.

