Heroku Deployment

For Homework 15 – Bellybutton Diversity

Steps

1. Prepare your application with additional configuration files
2. Create Heroku application
3. Prepare Heroku database

Configuration Files

Create a new conda virtual environment and install all dependencies in that environment. Learn more about conda environments.

1. Open Terminal or Anaconda Prompt. Make sure conda is up to date. From any directory, type

   ```
   conda update conda
   ```

   If your conda needs updating, you will be asked to confirm whether you wish to proceed. (Yes, you do!)

2. Create a new virtual environment with the conda create command. You will again be asked mid-way through confirm that you wish to proceed.
   - *The `-n` option indicates that what follows is the name of the virtual environment.*
   - *The `python=` syntax specifies the version of python that will be installed in this environment.*[1]

   ```
   conda create -n bellybutton_env python=3.6
   ```

3. "Enter" the virtual environment by activating it. It does not matter what directory you are in, because virtual environments are not based on your current location the way that git version control is. When you are in the environment, any python script you run from anywhere will use the version of python you set up in the active environment.[2] Once you've activated an env, you'll see the name of the environment in parenthesis before each command prompt (e.g., `(bellybutton_env) C:\Users\Claudia>`)

   Windows: `conda activate bellybutton_env`
   Mac: `source activate bellybutton_env`

4. Install dependencies – these are the packages you've been importing to run your Flask app. You'll add another, Gunicorn, which is a high performance web server that can run your Flask app in a production environment. You'll probably also need pandas, flask, flask-sqlalchemy, and any other libraries you are using that don't come already with a python installation.[3]

---

[1] *Later, when you enter this environment by activating it, this version of python will run any .py scripts. Conda will make sure that any additional packages you install in this environment will be compatible with this version of python.*

[2] *Jupyter is another story. If you are running a Jupyter Notebook, you have to do some other stuff to get it to recognize that you have virtual environments. Don't expect Jupyter to play nice with virtual environments on its own.*

[3] *When I tried installing my first dependency, I got an error that 'conda is not a recognized command.' I typed exit to close the anaconda prompt, then opened it again, activated my environment, and it worked. When you open a command prompt window, it reads environment variables as it starts up. If they have changed while*

```
conda install pandas
conda install flask
conda install sqlalchemy
conda install gunicorn
```

Uh-oh. What if conda can't find a package? Conda couldn't find gunicorn, even on the conda-forge channel. Not all packages are available through conda, so you might have to use pip.

It's unclear how conda and pip interact with each other. But if you go ahead and use pip to install gunicorn and flask-sqlalchemy (another conda no-find), it should work.

```
pip install gunicorn
pip install flask-sqlalchemy
```

Do you wonder where these packages are getting installed? When you have an activated environment, conda installs packages and python in the Anaconda3/envs directory. That's why it doesn't matter what directory you are in when you use conda commands. 'Conda install' will always install in its envs directory. Activating a virtual environment involves "pointing" to the right directory inside envs, so it does matter that you first activate your environment.

5. Test the app locally. ***Navigate to the directory that your app.py is in***, with the virtual environment is activated.

   Windows: `set FLASK_APP=app.py`
   Mac: `export FLASK_APP=app.py`
   then both: `flask run`
   Open a browser and go to http://localhost:5000

6. After confirming the app is running locally and that all the dependencies are installed, control-C to exit.

   Create a requirements.txt file.

   ```
   pip freeze > requirements.txt
   ```

7. Create a filed called Procfile (capital P, with no file extension). This file explicitly declares to Heroku what can be executed. It declares a web process type (ours is gunicorn) and declares the folder that contains the app as a python package.

   ```
   touch Procfile
   ```

   If this doesn't work, you can also just create a plain text file using your script editor.

   Type just one line in this file and save. This line does not mention any folder names. It assumes that your app.py file is in the same directory as Procfile.

   ```
   web: gunicorn app:app
   ```

8. Finish with git version control. Initialize a repository in this directory (you only need to do this once, not each time you change a file).

   ```
   git init
   ```

9. Add and commit all files.

   ```
   git add .
   git commit -m "First commit"
   ```

---

the window is open, (which may have happened with the conda update conda), then the variables need to be re-read by re-opening the command prompt.

Creating the Heroku App

Deploying code on Heroku is similar to doing a git push. How Heroku Works

1. First, you need an account on Heroku.
2. Use the "Create a New App" button to make your app. I chose Heroku Git for my deployment method. Instructions then appear under the selection. Follow them after installing Heroku CLI.
3. You'll have to install the Heroku CLI (a link is provided).
4. Create your Heroku app in the command line. This creates a new app on the Heroku servers.

   ```
   heroku create
   ```

   You can test that the new app was made with this command, which opens a browser with a placeholder for the app.

   ```
   heroku open
   ```

5. Push the committed files to the Heroku server.

   ```
   git push Heroku master
   ```

6. You should see this come up

   ```
   remote: -----> Python app detected
   ```

   Afterwards, you will see the dependencies from your requirements.txt file being installed with pip on the Heroku app. If there are errors about not finding a dependency from your requirements.txt file, you can remove that line from the file. [4]

   a. open your requirements.txt file
   b. remove the line that git push complained about, save.
   c. add, commit, and push the repo again.

7. Open a new Command Prompt, Terminal, or Anaconda Prompt. Log in to heroku. You'll be redirected to a browser then go back to Terminal to continue.

   ```
   heroku login
   ```

8. Start your app

   ```
   heroku open
   ```

   Hopefully you now see your app deployed on the web!

   If you get an error, follow the advice to close the tab and check the end of the log file. Read through it to see what the error is and try to address it.

   ```
   heroku logs --tail
   ```

Submitting the Link to Your App

In the Heroku website, on the Settings page for your app, scroll down to Domains and Certificates. There you will see a link to the app that can be submitted to BCS.

[4] *I ended up having to remove two of the dependencies that were installed along with pandas.*