Queen Mary
University of London

ECS781P: Cloud Computing Lab Instructions for Week 9

More steps towards app development on a PaaS

Dr. Arman Khouzani

March 7, 2017

Discovering the power of available APIs

1. There are many powerful API's around that provide advanced AI functionalities, audio and visual analysis, natural language processing functionalities. The basic usage is often provided free of charge as a means of promotion (in case your innovative app starts to gain attraction and you reach the limit of your free-account capacity, then you will need to upgrade to the priced grade). This is fantastic for prototyping and coming up with the next groundbreaking app or web mash-up.

   Most of these not only allow you to use their API, but also provide their functionalities as a Software Development Kit as well, with clear documentations and extensive examples. Here is a list of some of the most powerful APIs in artificial intelligence and natural language processing with a free-grade version:

   - IBM Watson (also allows development and deployment on IBM's cloud infrastructure called `Bluemix`): "Enable cognitive computing features in your app using IBM Watson's Language, Vision, Speech and Data APIs."
   - Google's Cloud Natural Language API (also can be used from within `Google Cloud Platform`): "Google Cloud Natural Language API reveals the structure and meaning of text by offering powerful machine learning models in an easy to use REST API. You can use it to extract information about people, places, events and much more, mentioned in text documents, news articles or blog posts."
   - Wit.ai: "Wit.ai makes it easy for developers to build applications and devices that you can talk or text to."
   - BigML: "Build Real-time Predictive Apps".
   - Diffbot("Artificial intelligence that are mobile first"

- ▶ The list will be too long, some others to mention: Amazon's Machine Learning, MindMeld, The Veles, PredictionIO, and even right near us in London's Tech City: TextRazor, ...

2. Today, we will be using one very simple Natural Language Processing API in order to make a simple interactive app. The app does the following: receives a sentence from the user, and based on the "sentiment" (mood) of the sentence, it responds accordingly.

3. The primary goal of the exercise is to see how one can use these powerful APIs and create an interactive web-based application. Specially, how using a PaaS makes deploying the apps simple.

4. Please make sure you have a running app by the end of today, so that we can solve the exercise of the second lab quiz, which will be posted during the week (deadline will be at March 31, 2017.)

First Steps Toward our Sentient Robot!

1. Open a terminal and enter your app (from last week) directory. Make sure you have the following file structure in place:

```
.git\
.openshift\
.gitignore
data\
lib\
setup.py
wsgi\
   application
   run.py
   tmp\
   venv\
   app\
     __init__.py
     views.py
     template\
     static\
```

   As a quick review:

   - ▷ `setup.py` is a file that tells the openshift platform which dependencies to install/use for deploying our app.
   - ▷ `.gitignore` is where we specified which files to be ignored by `git` and hence don't get added and pushed to the platform (these were e.g. compiled files for our local tests, and our local python packages installed in our virtual environment).
   - ▷ wsgi folder (stands for web server gateway interface) is where our code that generates the (dynamic) html pages reside.
   - ▷ `run.py` file runs our app (serves it on the flask web server).

- ▷ `application` is an open-shift specific file that tells the openshift where to find our app.
- ▷ tmp folder is where we keep temporary files.
- ▷ venv contained our isolated python libraries that we installed for local testing of the app before deploying it.
- ▷ app folder contained our app as a python package.
- ▷ `__init__.py` is a standard name file that specifies that this folder is a python package (contains modules) and may also execute some initializations (in our case, it created an instance of an object from the Flask class, and imported the `views` module).
- ▷ `views.py` file that creates our "views" (the dynamic html pages).
- ▷ template folder that contains our html "templates".
- ▷ static folder that contains our static files (like CSS, JS, Media files like Images, Videos, Audios, etc. that are used in the html pages)

2. In particular, if you have any other files, you can safely delete them by running `rm -r <name-of-the-folder>` for folders, and `rm <name-of-the-file>` for files.

3. First, we are going to make our website a bit better looking, with a combination of html templates, CSS and JS files. But we are not going to reinvent the wheel. We will be using a popular front-end package called Materialize. Note: if you would prefer, you can also use Bootstrap. The steps are almost identical. Got to your tmp folder and download the

   ```
   wget http://materializecss.com/bin/materialize-v0.98.0.zip
   ```

   Extracts its content (unzip it):

   ```
   unzip materialize-v0.98.0.zip
   ```

   Then move its entire content to our static folder:

   ```
   mv materialize/* ../app/static/
   ```

   Check if everything is done properly (check the contents of the static folder).

4. From QMPlus, download the HTML templates files named `base.html`, `index.html` and `my_form.html` and put them in our templates folder.

5. Also download the files `views.py`, `forms.py` and `__init__.py` and save them in the app folder (over-writing the previous files).

6. Activate your python virtual environment
   (recall: run `. <path-to-your-virtual-env-folder>/bin/activate`, don't forget the dot at the beginning!).

7. Install the necessary python libraries that you need for running the app locally (note: you can easily check that by investigating the libraries that are imported in our python files.)

8. Run the app locally (`python run.py`). Does the app work? What is it doing?

9. The app is missing 3 figures to be complete. You can investigate this by looking at the content of the `myform_html` and `views.py` files. Create a folder called img inside static and put your appropriate choice of 3 figures inside it (with proper naming that you find out from the code!). Re-run the app, does it work now?

10. Now edit the `setup.py` file to add the additional required packages. Deploy the app on `openshift`. Does it work?