Queen Mary
University of London

ECS781P: Cloud Computing
Lab Instructions for Week 11

Adding a (Mongodb) Database

Dr. Arman Khouzani

March 21, 2017

Adding a (Mongodb) Database

MongoDB is an open-source document-oriented NoSQL database program. Each database is composed of multiple collections and each collection has many documents. Each document has an automatically created unique id. A collection of documents has a similar format to an array of dictionaries, similar to the JSON formatting. Each document has many fields, which are similar to the "keys" of a dictionary.

MongoDB supports field, range and regular expression queries. It provides high availability through replica sets (redundant copies) using an active-passive failover system. It also supports load-balancing and an active-active failover system (using "sharding") and horizontal scaling. It also supports parallel distributed computing using the MapReduce framework.

Here, we will add a MongoDB database to our app, populate it with data and perform some simple queries on it.

1. First, let us add a MongoDB database using our `openshift` PaaS. (To see a list of available services that you can add, run `rhc cartridges`).

   ```
   rhc cartridge add mongodb-2.4 -a <name_of_your_app>
   ```

   Note the final prompt: the PaaS has automatically set up the database for us and provided the root user, password, and the name of the database (which is by default the same as the app name). Also, it tells us what url to use to connect to our data base:

   ```
   mongodb://$OPENSHIFT_MONGODB_DB_HOST:$OPENSHIFT_MONGODB_DB_PORT/
   ```

Note how the exact values are not returned, but only the name of their `environment` variables. This has two advantages: (1) These values can be changed parametrically; (2) we don't have to pass these values "in the clear" (which for some values, like passwords or tokens, is a major security flaw). To see a full list of environment variables you can issue:

```
rhc ssh <name_of_your_app> env
```

You can also add your own environment variables per each app by issuing:

```
rhc env set -a <name_of_your_app> VARIABLE=VALUE
```

The full url to connect to our MongoDB is composed of (all in one line):

```
mongodb://$OPENSHIFT_MONGODB_DB_USERNAME:
$OPENSHIFT_MONGODB_DB_PASSWORD@
$OPENSHIFT_MONGODB_DB_HOST:$OPENSHIFT_MONGODB_DB_PORT
```

This is all made available in a single environment variable for us called `OPENSHIFT_MONGODB_DB_URL`, which, we will use.

2. Python can connect to a MongoDB using many packages and modules like `pymongo`.[1] We will be using `flask`'s wrapper for `pymongo`. You can also access OS's environment variables using the `os` module, which provides an interface with the operating system. Create a python file inside your app folder – the same folder where your `__init__.py` and `views.py` module reside (call it e.g. `database.py`). At the beginning of the file, add:

```
from app import app
from flask_pymongo import PyMongo
from flask import jsonify
import os
```

By now you know the drill, you need to activate your virtual environment locally – making sure that the prompt changes to reflect this, then run
`pip install flask_pymongo`
Also, add `Flask-PyMongo` to our `install_requires` in our `setup.py` file.

3. Then add the following lines to the `database.py`:

```
app.config['MONGO_DBNAME'] = os.environ['OPENSHIFT_APP_NAME']
app.config['MONGO_URI'] = os.environ['OPENSHIFT_MONGODB_DB_URL']

mongo = PyMongo(app)
```

Note: if you are just copy-pasting, then fix the apostrophe and underlines!)

4. The mongo is now an "pymongo" object (client connection) object that allows you to create, delete, view, update collections and documents inside collections. Let us first write an API that shows the inside of our database (all the collections). Add the following to the database.py file:

[1] A short turotial can be found here: http://api.mongodb.com/python/current/tutorial.html.

```
@app.route('/database/collections', methods=['GET'])
def get_all_databases():
return jsonify({'result' : mongo.db.collection_names()})
```

5. To test the app locally, open a different terminal and run `rhc port-forward <name_of_your_app>`. Then export the appropriate variable.

6. Populate your data-base with some interesting documents, and show them through an api. The following code should work as a template (assuming a collection name `test` is created which has fields “Name” and ”Profession”):

```
@app.route('/database/personnel', methods=['GET'])
def get_all_personnel():
collection = mongo.db.test
output = []
for doc in collection.find():
    output.append({'Who' : doc['Name'], 'Job Role' : doc['Profession']})
return jsonify({'result' : output})
```