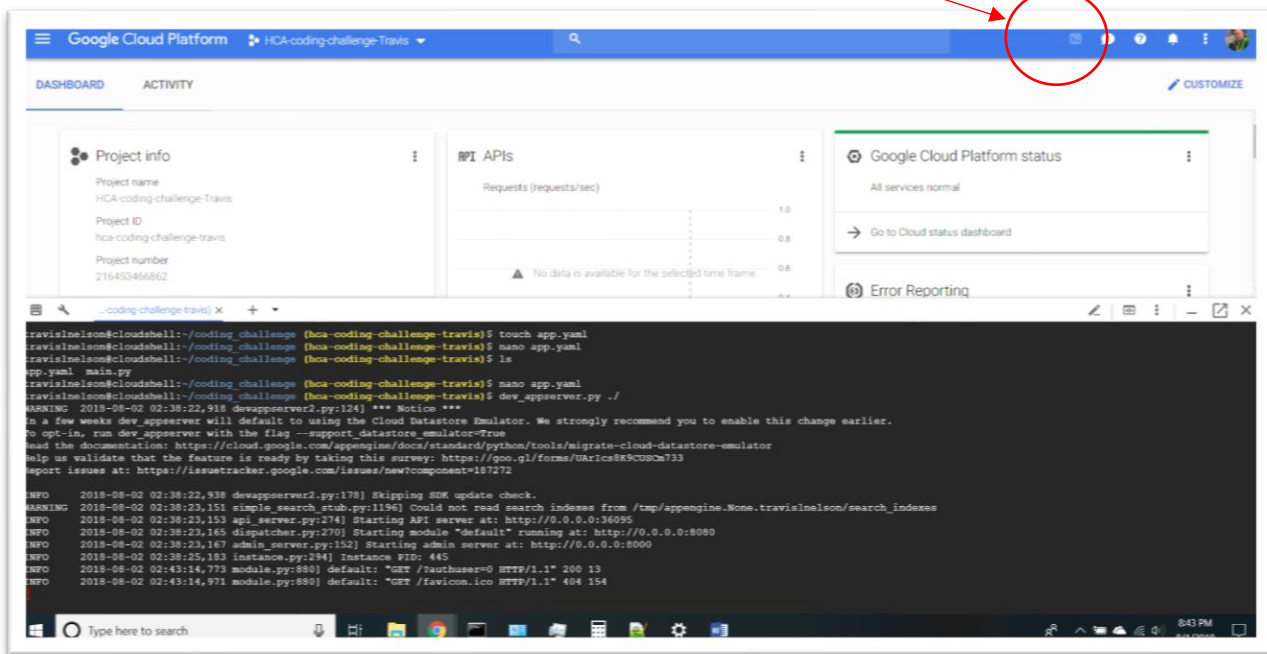


This document describes how to create a simple Web App for the HCA coding challenge. I chose to make a Web App using Python in Google App Engine, which provides both the frontend and backend of the application all in one convenient package. The backend is a Google Cloud Datastore, which is a NoSQL database.

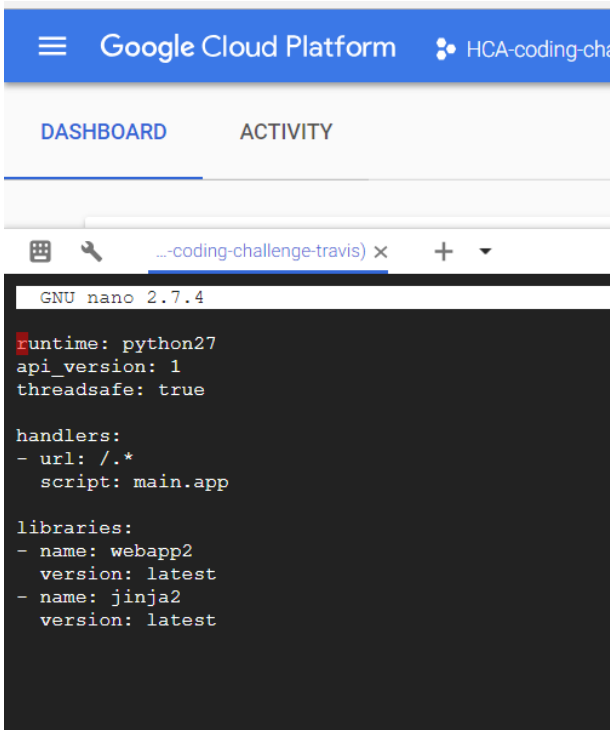
The url for this Web App is: <https://hca-coding-challenge-travis.appspot.com>

Setup:

1. Create a Google account if you don't already have one: <https://accounts.google.com/SignUp>
2. Once you have a Google account, go to their cloud platform: console.cloud.google.com and create a new project
3. Start a shell by clicking on the button in the toolbar:



4. The shell is for a Debian-based virtual machine that already has all of the development tools. So convenient!
5. From the command line, create the files: main.app and app.yaml
 - a. main.app is where we'll put the code for the project
 - b. app.yaml is needed for the webapp to work properly
 - c. We'll need some HTML files, I called mine index.html and results_page.html
6. In app.yaml, we just need some standard code. Jinja is a library that's used to render HTML that's stored in a separate file.



The screenshot shows the Google Cloud Platform console with the 'HCA-coding-challenge-travis' project selected. The 'DASHBOARD' tab is active. A terminal window is open, displaying the contents of the `app.yaml` file in the `nano` editor. The file content is as follows:

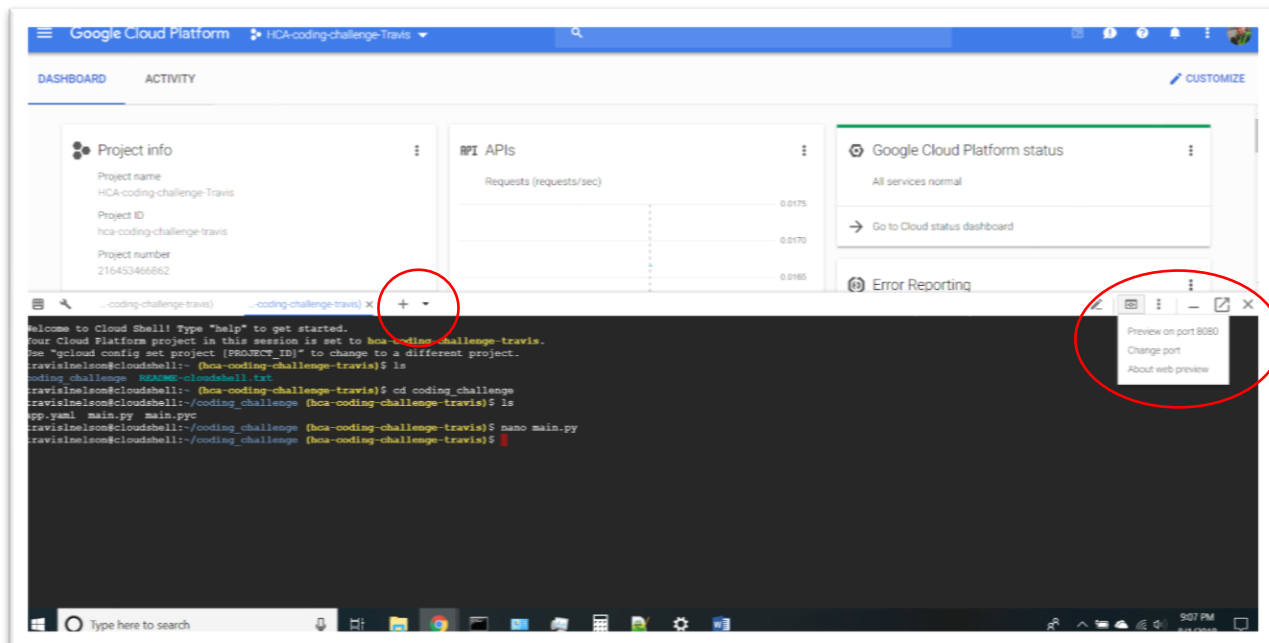
```
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /*.*
  script: main.app

libraries:
- name: webapp2
  version: latest
- name: jinja2
  version: latest
```

app.yaml

7. Eventually, we'll need to test the code, to do that, start up a development server with the command:
`dev_appserver.py ./`
8. While the first command shell is running the development server, you can open a second shell to do the rest of your development activities. Click on the + to start a new shell.
9. To view your test page, click on the Web Preview button and select 'Preview on Port 8080':



10. To edit the files, you can use nano, vim, or emacs. There's also a beta version of a document-style code editor, but I had problems with Python spacing using it. It might work better for other languages.

The main.app code:

1. First, the imports and setting up the Jinja environment:

```
import webapp2
import jinja2
import os
import urllib
import csv
from google.appengine.ext import ndb

#Sets up the Jinja environment, so we can pull in HTML code from a separate file
JINJA_ENVIRONMENT = jinja2.Environment(
    loader=jinja2.FileSystemLoader(os.path.dirname(__file__)),
    extensions=['jinja2.ext.autoescape'],
    autoescape=True)

...
```

main.py

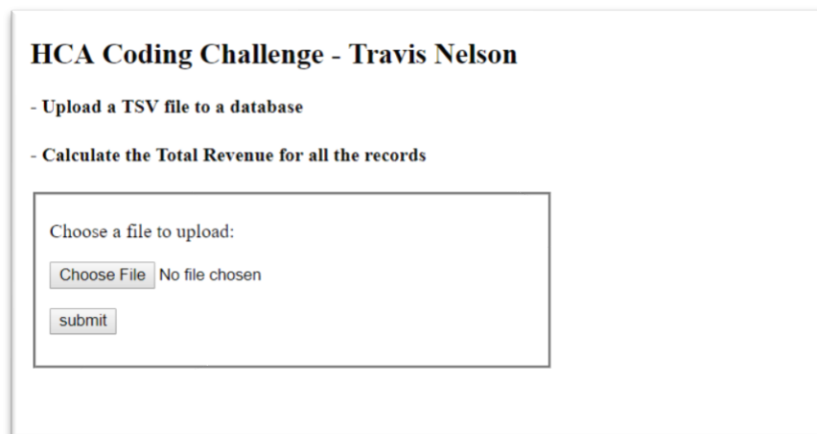
2. Each page of the Webapp is coded as a separate class, the first one is for the main page. All it does is to basically display the form for the user to upload a file. The HTML is stored in the index.html file, so the code doesn't get too messy:

```
...
# Right now, it's just a form for them to upload a file with
...
class MainPage(webapp2.RequestHandler):
    def get(self):
        #The HTML is kept in a separate file, to help keep the code organized
        template = JINJA_ENVIRONMENT.get_template('index.html')
        self.response.write(template.render())
...

```

main.py

3. The first page the user sees is just the form to upload a file. Obviously it's nothing fancy! There's a check on the HTML form to only accept *.tsv files.

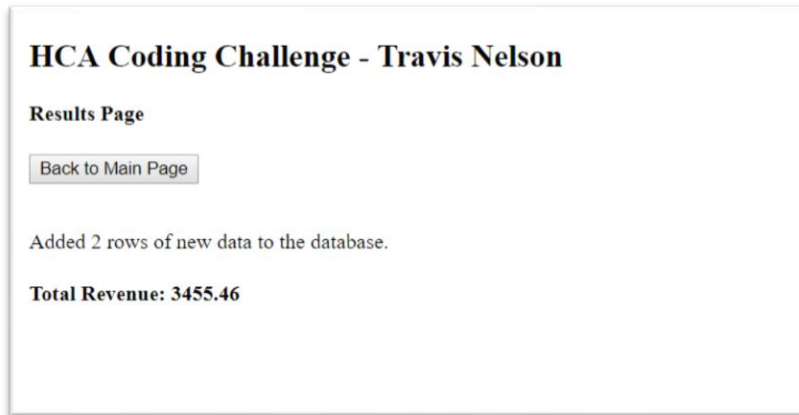


The screenshot shows a web browser window with the title "HCA Coding Challenge - Travis Nelson". Below the title, there are two bullet points: "- Upload a TSV file to a database" and "- Calculate the Total Revenue for all the records". Below these instructions is a form box. Inside the form box, it says "Choose a file to upload:". There are two buttons: "Choose File" and "submit". The "Choose File" button is disabled, and the text "No file chosen" is displayed next to it.

main HTML page

4. After hitting the submit button, a POST request is sent to the next page. If the user selects a file that's already in the database, the Python code will delete the old file and all of its data and then upload the new file. Otherwise,

it displays a little message telling the user how many rows of data were uploaded and the total revenue in the file, plus a little button to get back to the main page. Again, nothing fancy.



Results HTML Page

5. The code for the results page is in its own class, and the request is handled through the post method. The HTML is again stored in a separate file, then we just pull the uploaded file out of the POST request, run it through Python's csv reader to read it as a TSV file (by specifying the delimiter as '\t')

```
'''
class FileProcessing(webapp2.RequestHandler):
    #In case the location of the data changes, the index in the row where it's located can be easily changed here
    ITEM_PRICE_INDEX = 2
    ITEM_COUNT_INDEX = 3

    #After submitting the form to upload a file, a POST request is sent to this page, this is where that request goes
    def post(self):
        #The HTML for the page is stored in a separate file
        template_results = JINJA_ENVIRONMENT.get_template('results_page.html')
        self.response.write(template_results.render())

        #https://blog.whiteoat.com.uk/2016/08/01/handling-uploads-with-app-engine-and-webapp2/
        #Grabbing the file from the POST request
        raw_file = self.request.POST.get('tsv_file')

        #Error checking to make sure the file is a tsv file
        if not raw_file.filename.endswith('.tsv'):
            self.response.write("Uploaded file is not a tsv file!")
            return

        #Once we have the file, I'm using python's csv module to read it as a tsv file
        try:
            csv_reader = csv.reader(raw_file.file, delimiter='\t')
        except:
            self.response.write("Error reading the tsv file!")
            return

        #If the user tries to upload a file that's already in the database, this method will remove the old data so it can be replaced with the new stuff
        row_delete_count = self.delete_duplicate_data(raw_file)
```

main.py

6. The data is stored in Google's Cloud Datastore, a NoSQL document database. They call a piece of data an 'entity', and for this project I defined the name of the file as the parent and each row of data in the file is its child. To connect to the datastore, you use Google's NDB client library. The data doesn't have to be stored like this, the Datastore lets you name fields on the fly, but for structured data this works just fine. Each entity is defined by a class, and the properties of the entity are defined within the class

```

'''
This class is how the file information is structured and saved in the database
'''
class File_Name(ndb.Model):
    name = ndb.StringProperty()
    date = ndb.DateTimeProperty(auto_now_add=True)

'''
This class is how each individual row of data is structured and saved in the database
'''
class Data_Row(ndb.Model):
    item = ndb.StringProperty()
    description = ndb.StringProperty()
    price = ndb.StringProperty()
    count = ndb.StringProperty()
    vendor = ndb.StringProperty()
    vendor_address = ndb.StringProperty()

```

main.py

- To add an entity to the Datastore, you create an object of whatever kind you are storing, fill out the property fields, and use `.put()` to save it to the Datastore, the return value is an object that contains the unique key for the item in the datastore. The parent can be specified by its key, in this case the parent was created just before this for loop, so the key was saved and used here:

```

#The csv module should have converted each line of data in the tsv file into a list
for row in csv_reader:
    #The first row of data in the file is a header file, so we just skip it
    if first_row == True:
        first_row = False
        continue

    #Otherwise, we take the price and count data and multiply it, and then put the row of data into the database
    #If there are any errors, the except block will catch them and display it for the user
    try:
        total += float(row[self.ITEM_PRICE_INDEX]) * float(row[self.ITEM_COUNT_INDEX])
        if file_key is not None:
            row_key = Data_Row(item=row[0],description=row[1],price=row[2],count=row[3],vendor=row[4],vendor_address=row[5], parent=file_key.key)
            row_key.put()
            row_count += 1
        else:
            self.response.write("File key was none.")
    except Exception as e:
        row_errors += 1
        self.response.write("except: {}".format(e))

```

main.py

- The `self.response.write("--")` statements just output a line of text to the HTML page
- After code has been added or changed, the output can be easily seen by refreshing the port 8080 preview page
- When everything is finished, deploying the app is as simple as running the following command:

```
$ gcloud app deploy app.yaml
```

```

travisnelson@cloudshell: /coding_challenge (hca-coding-challenge-travis) $ gcloud app deploy app.yaml
Services to deploy:

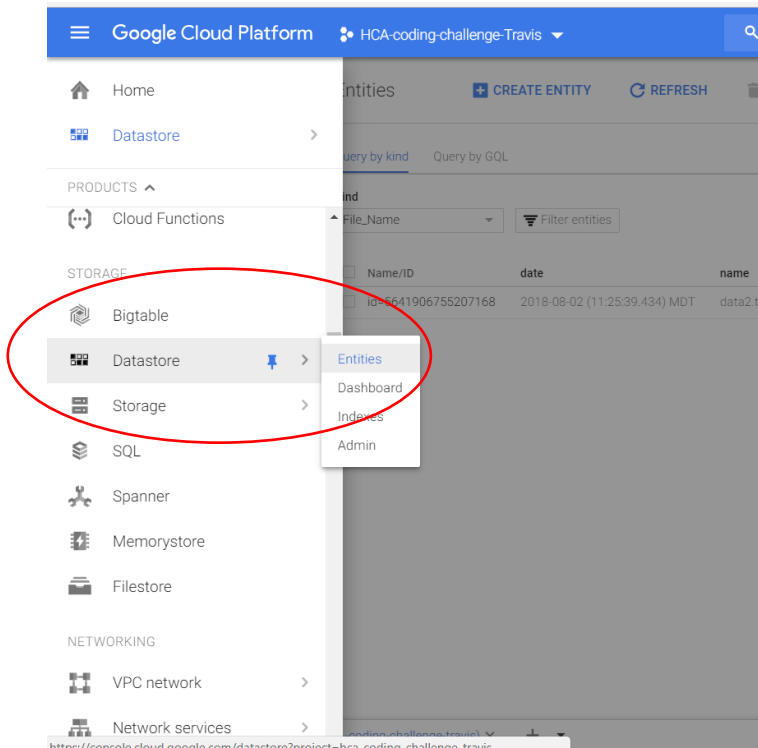
descriptor:    [/home/travisnelson/coding_challenge/app.yaml]
source:        [/home/travisnelson/coding_challenge]
target project: [hca-coding-challenge-travis]
target service: [default]
target version: [20180802t111132]
target url:    [https://hca-coding-challenge-travis.appspot.com]

Do you want to continue (Y/n)? 

```

Cloud Shell

- After it has been deployed, the data can be viewed in the Dashboard for the project. Click on the menu in the upper left hand corner, then scroll down to the Datastore



Google Cloud Platform HCA-coding-challenge-Travis

Entities CREATE ENTITY REFRESH DELETE

Query by kind Query by SQL

Kind Data_Row Filter entities

Number of columns to display 50

Name/ID	Parent	count	description	item	price
id=5066549580791808	Key('File_Name', 5631986051842048)	9	Fluoroscopy of Bilateral Kidneys using Other Contrast	Dried Peach	97.0
id=5068993417183232	Key('File_Name', 5631986051842048)	8	Transplantation of Nervous System into POC, Perc Approach	Squash - Pattypan, Yellow	34.01
id=5071437253574656	Key('File_Name', 5631986051842048)	33	Inspection of Left Wrist Joint, External Approach	Pork - Hock And Feet Attached	82.38
id=5073520312713216	Key('File_Name', 5631986051842048)	46	Repair Left Tarsal Joint, External Approach	Pepper - Scotch Bonnet	18.72
id=5076324926357504	Key('File_Name', 5631986051842048)	30	Excision of Pons, Percutaneous Endoscopic Approach	Egg Patty Fried	53.4
id=5078382215692288	Key('File_Name', 5631986051842048)	8	Excision of Left Maxillary Sinus, Perc Endo Approach, Diagn	Island Oasis - Wildberry	27.18
id=5080491044634624	Key('File_Name', 5631986051842048)	27	CT Scan of Paranasal Sinus using H Osm Contrast	Beans - Fava, Canned	70.84
id=5086100271923200	Key('File_Name', 5631986051842048)	28	Destruction of Endometrium, Open Approach	Spring Roll Veg Mini	93.72
id=5090214850592768	Key('File_Name', 5631986051842048)	13	Bypass Right Vas Deferens to L Vas Def, Perc Endo Approach	Olives - Nicoise	45.54
id=5094432508477440	Key('File_Name', 5631986051842048)	21	Revision of Autol Sub in R Sternoclav Jt, Extern Approach	Pork - Loin, Boneless	80.62
id=5096695956242432	Key('File_Name', 5631986051842048)	26	Lower Joints, Revision	Towel - Roll White	51.86
id=5102167744577536	Key('File_Name', 5631986051842048)	5	Revision of Autol Sub in Sternum, Perc Approach	Wine - Lamancha Do Crianza	1.55
id=5105650963054592	Key('File_Name', 5631986051842048)	25	Reposition Abdominal Aorta, Open Approach	Ecolab - Ster Bac	45.76

12. That's it. Enjoy and have fun!