**Google Cloud Platform – Professional Cloud Architect**

**(Lab commands & screenshots)**

App Engine, Cloud Functions, Cloud Run

**Objective: - Use the Google Cloud Translation API with Python and deploy to a Cloud serverless compute platform (App Engine, Cloud Functions, or Cloud Run).**

Steps:

1. Enable Cloud API’s (command line)

- gcloud services enable SERVICE\_NAME.googleapis.com

* gcloud services enable appengine.googleapis.com
* gcloud services enable artifactregistry.googleapis.com run.googleapis.com translate.googleapis.com

From GUI:

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

1. Clone the sample code

* git clone <https://github.com/googlecodelabs/cloud-nebulous-serverless-python.git>
* Graphical user interface, text, website

  Description automatically generated

1. Deploy the service
   1. App Engine
      1. Modify app.yaml

Uncomment runtime: python38 and delete everything else.

* + 1. Deploy the app – gcloud app deploy
    2. Browse the app – gcloud app browse
  1. Cloud Function
     1. gcloud functions deploy translate --runtime python37 --trigger-http --allow-unauthenticated
     2. Application URL:

https://*REGION*-*PROJECT\_ID*.cloudfunctions.net/translate

* 1. Cloud Run
     1. Rename docker file to .dockerfile to avoid hindrance.
     2. gcloud run deploy translate --source . --allow-unauthenticated --platform managed

Appendix:

Code Explanation:

The sample app is a simple [Google Translate](http://translate.google.com/) derivative that prompts users to enter text in English and receive the equivalent translation of that text in Spanish. Now open the main.py file so we can see how it works. Omitting the commented lines about licensing, it looks like this at the top and bottom:

from flask import Flask, render\_template, request  
import google.auth  
from google.cloud import translate  
  
app = Flask(\_\_name\_\_)  
\_, PROJECT\_ID = google.auth.default()  
TRANSLATE = translate.TranslationServiceClient()  
PARENT = 'projects/{}'.format(PROJECT\_ID)  
SOURCE, TARGET = ('en', 'English'), ('es', 'Spanish')  
  
# . . . [translate() function definition] . . .  
  
if \_\_name\_\_ == '\_\_main\_\_':  
    import os  
    app.run(debug=True, threaded=True, host='0.0.0.0',  
            port=int(os.environ.get('PORT', 8080)))

1. The imports bring in Flask functionality, the google.auth module, and the Cloud Translation API client library.
2. The global variables represent the Flask app, the Cloud project ID, the Translation API client, the parent "location path" for Translation API calls, and the source and target languages. In this case, it's English (en) and Spanish (es)
3. The large if block at the bottom is used in the tutorial for **running this app locally**—it utilizes the Flask development server to serve our app. This section is also here for the **Cloud Run deployment** tutorials in case the web server isn't bundled into the container. You are asked to enable bundling the server in the container, but in case you overlook this, the app code falls back to using the Flask development server. (It is not an issue with App Engine or Cloud Functions because those are sourced-based platforms, meaning Google Cloud provides and runs a default web server.)

Finally, in the middle of main.py is the heart of the application, the translate() function:

@app.route('/', methods=['GET', 'POST'])  
def translate(gcf\_request=None):  
    """  
    main handler - show form and possibly previous translation  
    """  
  
    # Flask Request object passed in for Cloud Functions  
    # (use gcf\_request for GCF but flask.request otherwise)  
    local\_request = gcf\_request if gcf\_request else request  
  
    # reset all variables (GET)  
    text = translated = None  
  
    # if there is data to process (POST)  
    if local\_request.method == 'POST':  
        text = local\_request.form['text']  
        data = {  
            'contents': [text],  
            'parent': PARENT,  
            'target\_language\_code': TARGET[0],  
        }  
        # handle older call for backwards-compatibility  
        try:  
            rsp = TRANSLATE.translate\_text(request=data)  
        except TypeError:  
            rsp = TRANSLATE.translate\_text(\*\*data)  
        translated = rsp.translations[0].translated\_text  
  
    # create context & render template  
    context = {  
        'orig':  {'text': text, 'lc': SOURCE},  
        'trans': {'text': translated, 'lc': TARGET},  
    }  
    return render\_template('index.html', \*\*context)

The primary function does the work of taking the user input, and calling the Translation API to do the heavy-lifting. Let's break it down:

1. Check to see if requests are coming from Cloud Functions using the local\_request variable. Cloud Functions sends in its own [Flask Request object](https://flask.palletsprojects.com/api/#flask.Request) whereas all others (running locally or deploying to App Engine or Cloud Run) will get the request object directly from Flask.
2. Reset the basic variables for the form. This is primarily for GET requests as POST requests will have data that replace these.
3. If it's a POST, grab the text to translate, and create a JSON structure representing the API metadata requirement. Then call the API, falling back to a previous version of the API if the user is employing an older library.
4. Regardless, format the actual results (POST) or no data (GET) into the template context and render.

The visual part of the application is in the template index.html file. It shows any previously translated results (blank otherwise) followed by the form asking for something to translate:

<!doctype html>  
<html><head><title>My Google Translate 1990s</title><body>  
<h2>My Google Translate (1990s edition)</h2>  
  
{% if trans['text'] %}  
    <h4>Previous translation</h4>  
    <li><b>Original</b>:   {{ orig['text'] }}  (<i>{{ orig['lc'][0] }}</i>)</li>  
    <li><b>Translated</b>: {{ trans['text'] }} (<i>{{ trans['lc'][0] }}</i>)</li>  
{% endif %}  
  
<h4>Enter <i>{{ orig['lc'][1] }}</i> text to translate to <i>{{ trans['lc'][1] }}</i>:</h4>  
<form method="POST"><input name="text"><input type="submit"></form>  
</body></html>