

# Report

## Assignment 5

### Function as a Service

For this assignment, I implemented a text analyser with a simple web based UI. For this I used

- Google Cloud Functions in python3 to perform text analysis
- Google Cloud Function in python3 to create a matplotlib graph out of the analysis performed (**bonus**)
- Google Cloud Storage as cache (**bonus**)
- JavaScript and HTML5 for the UI and hosted it on homes.homes.sice.indiana.edu (**bonus**)

In the application, there is a drop down list having several options for text files that are links to books in Project Gutenberg. Below is the link to the application. You can test it for yourself:

<https://homes.luddy.indiana.edu/msureshk/TextAnalysis/>

**Note: No value is cached in storage. All cached graphs have been deleted for grading purposes.**

### Implementation

*Note:* the python, JS and HTML code files along with log files and other information are added to A5.zip. Please check. More details of these files are given below.

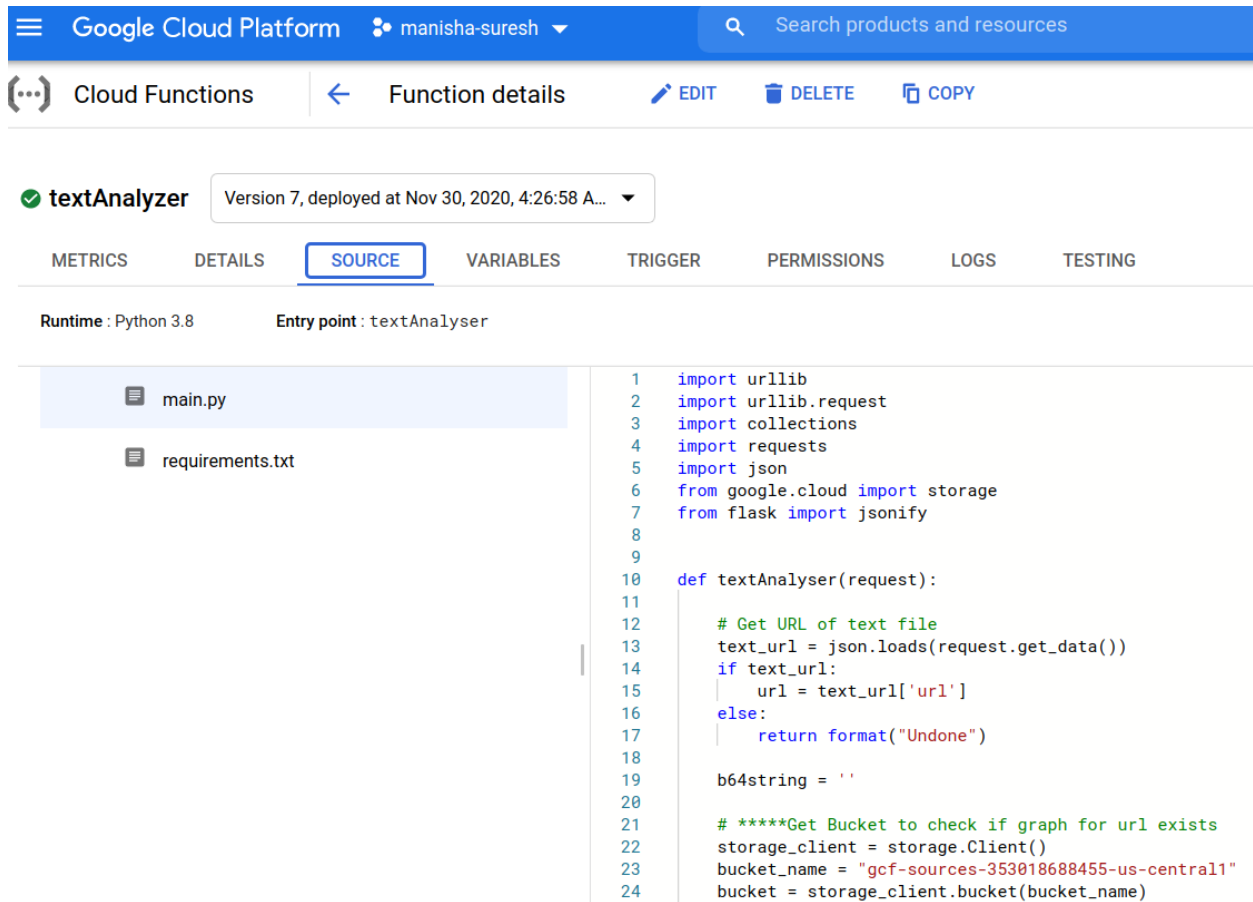
*High Level Implementation:*

1. User selects a text URL.
2. Javascript code sends XMLHttpRequest to the cloud function, 'textAnalyzer' passing the text URL as JSON in the request body.
3. textAnalyzer first checks if a graph exists for the requested URL in cache.
  - a. If yes, then fetch the graph from cache and returns as response
  - b. If no, then fetch contents of the URL. This will be text.
    - i. Split the text into sentences (search for periods) and create a distribution as a json file (Format : {<No. of words in sentence> : <count>})
    - ii. Send HTTP request to the cloud function, 'graph' and pass the name of JSON filename in the request body.
    - iii. graph fetches the json distribution from storage and using Matplotlib, generates a bar graph showing the distribution.
    - iv. Returns generated graph as response to textAnalyzer.
    - v. textAnalyzer caches this graph in cache for future requests
    - vi. Returns graph as response to JavaScript XMLHttpRequest

4. JS code will display the graph along with a link to text URL in the UI

#### Google Cloud Function:

My two functions , '*textAnalyzer*' and '*graph*' were deployed on Google Cloud Functions. Deployment was done with zip files containing a *main.py* file and a *requirements.txt* file containing dependencies that need to be installed. Below are screenshots of this for both functions



The screenshot displays the Google Cloud Platform console interface for a function named 'textAnalyzer'. The top navigation bar includes the Google Cloud Platform logo, the user name 'manisha-suresh', and a search bar. Below the navigation bar, the 'Cloud Functions' section is active, showing the 'Function details' page. The function 'textAnalyzer' is selected, with a dropdown menu indicating 'Version 7, deployed at Nov 30, 2020, 4:26:58 A...'. The 'SOURCE' tab is selected, showing the source code for the function. The code is written in Python 3.8 and defines a function 'textAnalyser(request)'. The code imports 'urllib', 'urllib.request', 'collections', 'requests', 'json', 'google.cloud.storage', and 'flask.jsonify'. It then defines the 'textAnalyser' function, which takes a 'request' object as input. The function first gets the URL of the text file from the request data. If the URL is present, it extracts it. Otherwise, it returns a message 'Undone'. The code then defines a 'b64string' variable and a comment indicating the next step is to get the bucket to check if the graph for the URL exists. It then creates a 'storage\_client' and a 'bucket\_name' variable, and finally creates a 'bucket' object using the 'storage\_client' and 'bucket\_name'.

```
1 import urllib
2 import urllib.request
3 import collections
4 import requests
5 import json
6 from google.cloud import storage
7 from flask import jsonify
8
9
10 def textAnalyser(request):
11
12     # Get URL of text file
13     text_url = json.loads(request.get_data())
14     if text_url:
15         url = text_url['url']
16     else:
17         return format("Undone")
18
19     b64string = ''
20
21     # *****Get Bucket to check if graph for url exists
22     storage_client = storage.Client()
23     bucket_name = "gcf-sources-353018688455-us-central1"
24     bucket = storage_client.bucket(bucket_name)
```

textAnalyzer

**textAnalyzer** (To see the *main.py* and *requirements.txt* files for *textAnalyzer*, check *textAnalyzis* folder inside *A5*) : This function performs analysis, calls on *graph* to create a visual representation of the analysis and returns the graphical figure as response to the front end script. There are three things to mention here

1. For each url, there will be only one file in the cache. Name of the file will be the url passed as part of the request. The file first holds the initial json distribution. After the graph is created, the contents are overwritten with the graph.

- For URLs that have already been served, the graphs are cached so that next time the same URL is sent. Since the name of the file is the same as the URL, I simply check for the file name in the cache and return the graph without needing any of the processing.
- The image is converted to a string of bytes using python's *base64* encoding and sent as response. To take a look at what this would look like, check '*PlotBytesInStringFormatExample.txt*'. This holds the graphical image of the distribution of [Pride and Prejudice](#) as a string of bytes.

The screenshot shows the Google Cloud Platform console interface. At the top, there's a navigation bar with the Google Cloud Platform logo, the user name 'manisha-suresh', and a search bar. Below this, the 'Cloud Functions' section is active, showing 'Function details' for a function named 'graph'. The function is version 4, deployed at Nov 30, 2020, 3:45:32 A... The tabs include METRICS, DETAILS, SOURCE (selected), VARIABLES, TRIGGER, PERMISSIONS, LOGS, and TESTING. The runtime is Python 3.8 and the entry point is graph. On the left, a file explorer shows 'requirements.txt' and 'main.py'. The main area displays the source code for 'main.py'.

```

1  from google.cloud import storage
2  from matplotlib import pyplot as plt
3  import json
4  import base64
5  from PIL import Image
6  import io
7  from flask import jsonify
8
9  def graph(request):
10
11     storage_client = storage.Client()
12     bucket_name = "gcf-sources-353018688455-us-central1"
13
14     json_name = json.loads(request.get_data())
15     source_blob_name = ''
16     if json_name:
17         source_blob_name = json_name['file_name']
18     else:
19         return format("Undone")
20
21     bucket = storage_client.bucket(bucket_name)
22     blob = bucket.blob(source_blob_name)
23
24     distribution_json = json.loads(blob.download_as_string())

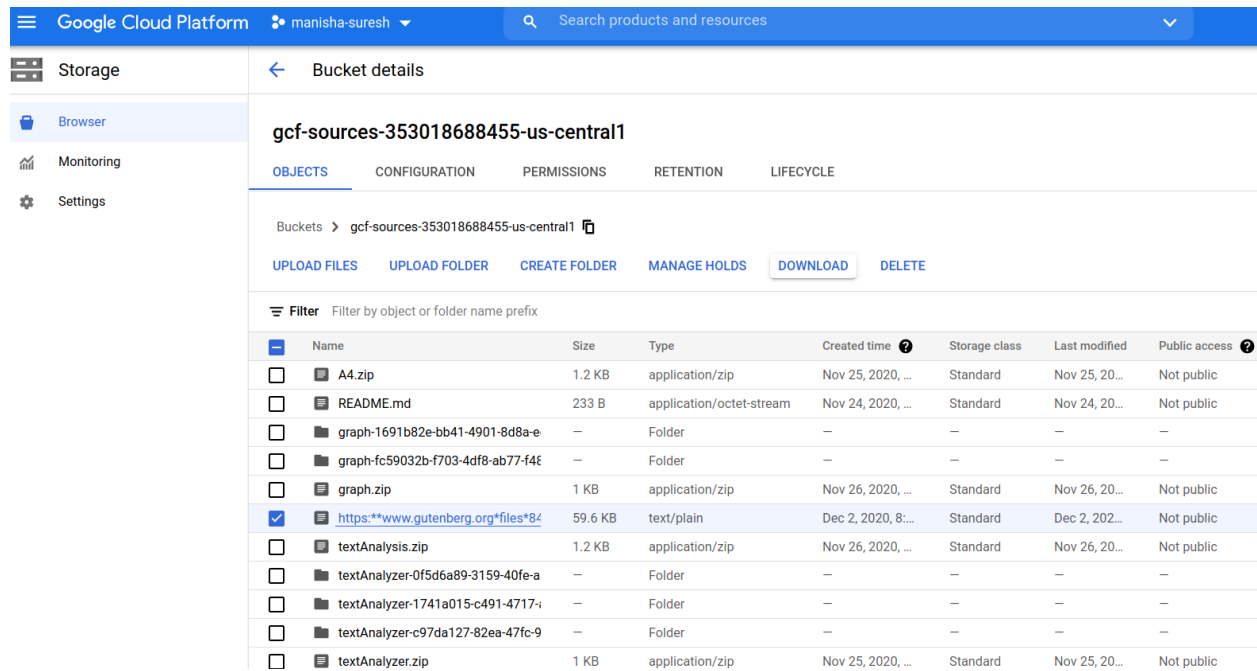
```

graph

**graph** (To see the *main.py* and *requirements.txt* files for *graph*, check *graph* folder inside A5) : *graph* is a fairly simple function that reads the distribution from cache. The distribution is in the form of a json file. To create a bar graph, the x-axis corresponds to the keys of the json file and y-axis corresponds to the values. The image to bytes conversion mentioned earlier takes place here before it is sent as response back to textAnalyzer. The images have a dimension of 30x15 and so, irrespective of the size of the json file, the size of response will be deterministic.

## Google Cloud Storage:

During cloud deployment, we need to select a storage bucket to hold the code files needed during cold restart. I made use of the same bucket for cache. Below is the screenshot of contents of the bucket.



Google Cloud Platform							
manisha-suresh							
Search products and resources							
Storage							
Bucket details							
gcf-sources-353018688455-us-central1							
OBJECTS CONFIGURATION PERMISSIONS RETENTION LIFECYCLE							
Buckets > gcf-sources-353018688455-us-central1							
UPLOAD FILES UPLOAD FOLDER CREATE FOLDER MANAGE HOLDS DOWNLOAD DELETE							
Filter Filter by object or folder name prefix							
	Name	Size	Type	Created time	Storage class	Last modified	Public access
<input type="checkbox"/>	A4.zip	1.2 KB	application/zip	Nov 25, 2020, ...	Standard	Nov 25, 20...	Not public
<input type="checkbox"/>	README.md	233 B	application/octet-stream	Nov 24, 2020, ...	Standard	Nov 24, 20...	Not public
<input type="checkbox"/>	graph-1691b82e-bb41-4901-8d8a-e	—	Folder	—	—	—	—
<input type="checkbox"/>	graph-fc59032b-f703-4df8-ab77-f4e	—	Folder	—	—	—	—
<input type="checkbox"/>	graph.zip	1 KB	application/zip	Nov 26, 2020, ...	Standard	Nov 26, 20...	Not public
<input checked="" type="checkbox"/>	https:*www.gutenberg.org*files*84	59.6 KB	text/plain	Dec 2, 2020, 8:...	Standard	Dec 2, 202...	Not public
<input type="checkbox"/>	textAnalysis.zip	1.2 KB	application/zip	Nov 26, 2020, ...	Standard	Nov 26, 20...	Not public
<input type="checkbox"/>	textAnalyzer-0f5d6a89-3159-40fe-a	—	Folder	—	—	—	—
<input type="checkbox"/>	textAnalyzer-1741a015-c491-4717-i	—	Folder	—	—	—	—
<input type="checkbox"/>	textAnalyzer-c97da127-82ea-47fc-9	—	Folder	—	—	—	—
<input type="checkbox"/>	textAnalyzer.zip	1 KB	application/zip	Nov 25, 2020, ...	Standard	Nov 25, 20...	Not public

In the figure, the selected file is a cache for a URL. In code, Google Cloud Storage API is used to upload, download and check if a file exists.

## Web UI:

**processSelection.js** (To see js code, see file processSelection.js under the folder UI inside A5): The main point to note here, is that once the response is received from textAnalyzer, it is in the form of a string of bytes which is converted to an image of png format and displayed onto the screen within a classic <img> HTML tag. This file along with the html file forms the UI and is what is hosted in homes.sice.indiana.edu

## Performance

End-to-end times (when not already in cache):

- Text URL : A Tale of Two Cities, by Charles Dickens (<https://www.gutenberg.org/files/98/>)

File size : 785K

Time : 19s

- Text URL : Metamorphosis, by Franz Kafka(<http://www.gutenberg.org/cache/epub/5200/>)

File size : 138K

Time : **10s**

*Approximate time for 'graph' to execute during cold start : 7-9 s*

*Approximate time for 'graph' to execute in warm instances : 3-5 s*

*Approximate time for cached images : 2-5 ms*

## Log Files and Example Graph Images

- *textAnalysis\_logs.csv*, *graph\_logs.csv* inside A5 contain logs for execution start, cold start and end of execution.
- There are two example json distributions and corresponding graphs for the below two files that can be found in examples folder in A5. the .png files are the graphs and .txt files contains the json.

jekyllhyde → <https://www.gutenberg.org/files/43/43-0.txt>

metamorphosis → <http://www.gutenberg.org/cache/epub/5200/pg5200.txt>

## Improvements

1. A better analysis algorithm can be implemented to speed up the process. Currently, the implementation first splits the text when periods are encountered and then iterates to each split to count the number of words. Using an inbuilt NLTK library will do this task much efficiently and accurately because splitting according to periods alone doesn't mean single sentences are captured. '?', '!' are also possible punctuations which the current implementation does not handle.
2. When the intermediate json distribution is generated, it is stored in the cache and then another function reads it from the cache to generate graphs. This could be unnecessary storage operations and the json distribution can be passed in the request from first to second function. However, this may not be scalable when the text is huge and distribution json can also be huge. A counter argument to this would be that a sentence typically would not have more than 20 words in which case the json distribution will have a maximum of only 20 key-value pairs.