Assignment - Lab 3

ME16B172 - Sushant Uttam Wadavkar

1. The screenshot for logs of all the 3 tasks containing the required result - line count using VM, Cloud Functions and DataFlow, are provided below along with the python file for each task attached in the zip file.

VM1&2

```
me16b172@instance-2:~$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from google.cloud import storage
>>> client = storage.client()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'module' object is not callable
>>> client = storage.Client()
>>> bucket = client.get_bucket('sushantw')
>>> blob = bucket.get_blob('addresses.csv')
>>> x = blob.download_as_string()
>>> x = x.decode('utf-8')
>>> x = x.split('\n')
>>> count = len(x)-1
>>> count
22
>>>
```

```
me16b172@instance-2:~/lab3$ python3
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> f =  open('addresses.csv', 'r')
>>> for line in f:
...
...
  File "<stdin>", line 3

    ^
IndentationError: expected an indented block
>>> f = open('addresses.csv', 'r')
>>> count = 0
>>> for line in f:
...     count += 1
...
>>> count
22
```

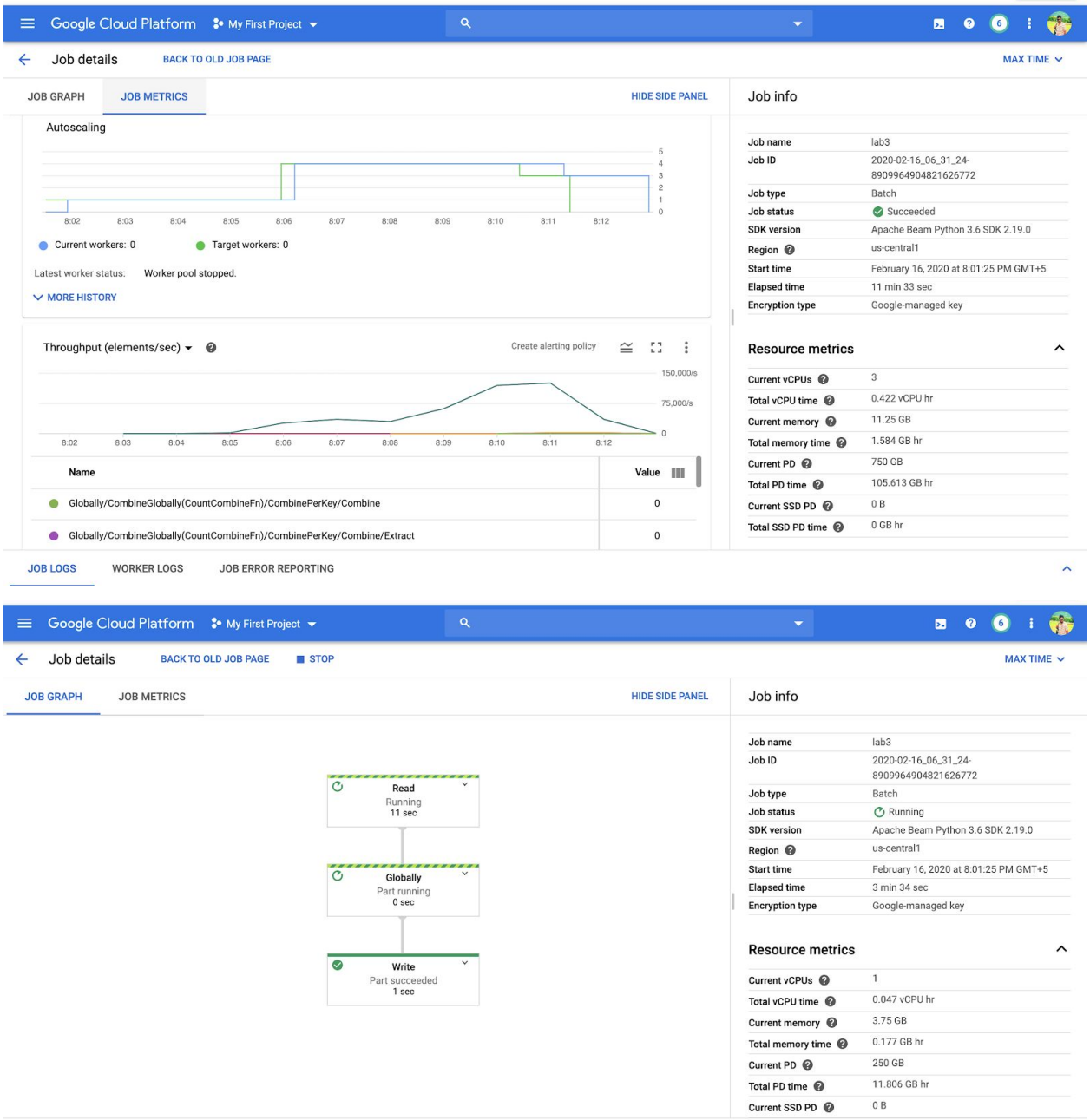
Cloud Function:

```
D      line_count  991922635793268  2020-02-16 14:41:24.249  Function execution took 1391 ms, finished with status: 'crash'
D      line_count  991913337703363  2020-02-16 14:41:26.491  Function execution started
I      line_count  991913337703363  2020-02-16 14:41:27.565  1
D      line_count  991913337703363  2020-02-16 14:41:27.569  Function execution took 1079 ms, finished with status: 'ok'
D      line_count  992033787579690  2020-02-16 16:26:42.892  Function execution started
E      line_count  992033787579690  2020-02-16 16:27:10.313  Error: memory limit exceeded. Function invocation was interrupted.
D      line_count  992039438871529  2020-02-16 16:28:19.345  Function execution started
E      line_count  992039438871529  2020-02-16 16:28:50.479  Error: memory limit exceeded. Function invocation was interrupted.
me16b172@instance-2:~/lab3$
```

Dataflow:

```
I      line_count  991872035661623  2020-02-16 13:59:42.938  22
D      line_count  991872035661623  2020-02-16 13:59:42.944  Function execution took 3715 ms, finished with status: 'ok'
D      line_count  991878886385080  2020-02-16 14:01:58.620  Function execution started
I      line_count  991878886385080  2020-02-16 14:02:01.647  22
D      line_count  991878886385080  2020-02-16 14:02:01.651  Function execution took 3033 ms, finished with status: 'ok'
me16b172@instance-2:~/lab3$
```

2. The screenshot for the execution graph created by Dataflow in the background for the pipeline object created in task 3 is being attached.



25974026

3. The PCollection abstraction represents a potentially distributed, multi-element data set. PCollection can be thought of as "pipeline" data; Beam transforms use PCollection objects as inputs and outputs. As such, to work on data in pipeline, it must be in the form of a PCollection. It takes input in the form of PCollection. We need to create a bucket in which output would be stored. We used the command beam.combiners.Count.Globally() to count the number of elements present in the given form of pipeline.

Issues faced and resolving them:

1. Confusion of reading and writing the files into my bucket; the problem of syntax understanding
    a. Careful reading of sample syntax available on stackoverflow
    b. Explanation given in the later class
2. Understanding the concept of pipeline (PCollection) was difficult through self-learning
    a. The explanation at the end of next week's lab was appropriate and helped in solving the specific doubts such as reading and writing the files into the bucket; how dataflow works in general

4. **Unbounded Data:**

The dataset in which for a particular time instance there are no bounds on the available data that is being fed to the system in this case to the pipeline, is called the unbounded data. We do not get access to entire data at once in the case of unbounded data.

PCollection application for Unbounded Data:

Windowing allows grouping operations over unbounded data collections by diving the collection into a window of finite collections according to the timestamps of the individual elements. A windowing function gave the way to assign elements to individual windows, and ways to merge those windows together. Apache beam permits us to outline different sorts of windows or the use of predefined windowing functions. For unbounded data the results are emitted when the watermark passes the end of the window indicating that the system believes all input data for the window has been processed. But there are no guarantees that data events can appear in the pipeline in the same order they were generated.