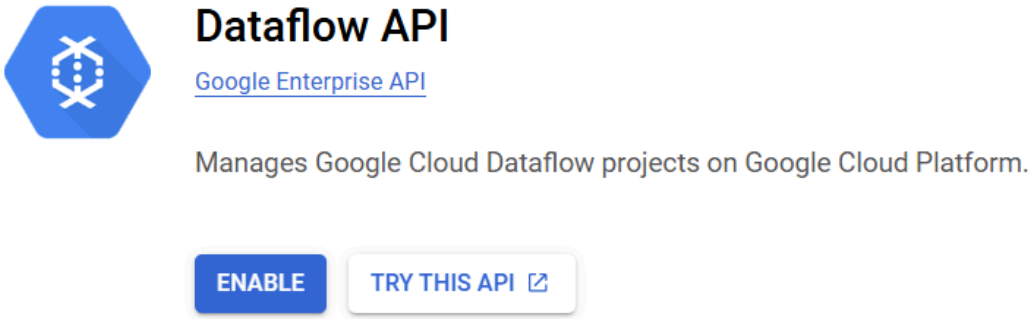
Name: Jeremy Bui
Course: DS 561

Homework 7

Setup

- Use the files directory inside the bucket from ps 2
- Pip install:
    - pip install apache-beam[gcp]
- gcloud auth application-default login
- gcloud config set project ds561-bucs
- gcloud config set compute/region us-central1
- gcloud projects add-iam-policy-binding ds561-bucs --member="serviceAccount:844994497823-compute@developer.gserviceaccount.com" --role="roles/dataflow.worker"
- Enable dataflow API



- To run call “python3 main.py”

Code

requirements.txt

```
apache-beam[gcp]==2.60.0
```

- Specify to install Apapache beam library with google cloud platform

main.py

```
import apache_beam as beam
```

```
from apache_beam.options.pipeline_options import PipelineOptions,
GoogleCloudOptions, WorkerOptions
from apache_beam.io import fileio
import time

class ExtractLinksFn(beam.DoFn):
    def process(self, file):
        page_name = file.metadata.path.split('/')[-1].split('.')[0]
        page_content = file.read_utf8().splitlines()
        outgoing_links = [
            line.split('"')[1].split('.')[0]
            for line in page_content
            if 'href' in line.lower() and '"' in line
        ]
        yield (page_name, outgoing_links)

class ComputeLinkStatistics(beam.DoFn):
    def process(self, element):
        page, links = element
        yield (page, len(links))

def run_pipeline():
    bucket_name = 'jeremybui_ps2'
    folder_name = 'files'

    job_name = f"apache-beam-dataflow-job-{int(time.time())}"

    # Set up PipelineOptions with DataflowRunner or DirectRunner
    options = PipelineOptions(
        runner='DataflowRunner',
        project='ds561-bucs',
        region='us-central1',
        job_name=job_name,
        staging_location=f'gs://{bucket_name}/staging',
        temp_location=f'gs://{bucket_name}/temp',
        requirements_file='requirements.txt'
    )

    google_cloud_options = options.view_as(GoogleCloudOptions)
```

```
    google_cloud_options.service_account_email =
'dataflow-service@ds561-bucs.iam.gserviceaccount.com'

    worker_options = options.view_as(WorkerOptions)
    worker_options.num_workers = 4

    start_time = time.time()

    with beam.Pipeline(options=options) as p:
        files = (p
                 | "Match Files" >>
beam.io.fileio.MatchFiles(f'gs://{bucket_name}/{folder_name}/*')
                 | "Read Matches" >> beam.io.fileio.ReadMatches())

        link_data = (files
                     | "Extract Links" >> beam.ParDo(ExtractLinksFn()))

        outgoing_counts = (link_data
                           | "Compute Outgoing Links" >>
beam.ParDo(ComputeLinkStatistics())
                           | "Combine Outgoing to List" >>
beam.combiners.ToList()
                           | "Top 5 Outgoing" >> beam.Map(lambda elements:
sorted(elements, key=lambda x: x[1], reverse=True)[:5]))

        incoming_counts = (link_data
                           | "Flatten Links" >> beam.FlatMap(lambda x:
[(link, 1) for link in x[1]])
                           | "Count Incoming Links" >>
beam.combiners.Count.PerKey()
                           | "Combine Incoming to List" >>
beam.combiners.ToList()
                           | "Top 5 Incoming" >> beam.Map(lambda elements:
sorted(elements, key=lambda x: x[1], reverse=True)[:5]))

        outgoing_counts | "Write Top Outgoing" >>
beam.io.WriteToText(f'gs://{bucket_name}/top_outgoing')
        incoming_counts | "Write Top Incoming" >>
beam.io.WriteToText(f'gs://{bucket_name}/top_incoming')
```

```
    end_time = time.time()
    runtime = end_time - start_time
    print(f"Pipeline runtime: {runtime:.2f} seconds")

if __name__ == "__main__":
    print("Running pipeline...")
    run_pipeline()
```

- Import Apache Bram library to define data processing pipelines
- Import classes to configure and set options for pipeline
- Import fileio to match and read files from cloud storage
- Import time to track execution time
- Logger = logging.getLogger(__name__) creates logger for the module
- Class ExtractLinksFn creates a custom function to process each file by extracting outgoing links and returning them
    - The process(self, file) method will operate on each file
    - page_name extracts the file name from the file's path to know the page identifier
    - page_content reads the content from the file and splits it into lines
    - outgoing_links finds the outgoing links by referring to href and splitting based on that
    - Yield then gets a tuple of the page-name and a list of each outgoing_link for each file and pass to next stage of pipeline
- ComputeLinkStatistics is a function to calculate number of outgoing links for each page
    - Process wil again be used for each element
    - We then want to unpack the page_name and outgoing_links
    - Yield to get the page name and count of outgoing links with len(links)
- run_pipeline() sets up the Apache Beam pipeline by giving the bucket name and folder name
- We can then create unique job with apache-beam-dataflow-job and the current time combined
- We then want to set u the pipeline options for the pipeline
    - Identify runner type (Direct runner to run locally or DataflowRunner to use google's cloud)
    - Project ID
    - Region for dataflow
    - Job_name (from what we declared before)

    - Staging_location to hold files required to run the job (dependencies / compiled pipeline)
    - temp_location to store temporary files during pipeline execution to write temp output as it processes data
    - requirements.txt to make sure the dataflow workers have all the python dependencies needed
- I then needed to do google_cloud_options as my permissions were not working earlier
- Moreover, I added more workers to speed up the process
- We can then do start_time = time.time() for the time to start
- To build the pipeline:
    - With beam.pipeline(options=options) as p create a new Apache Beam pipeline
    - Match files allow us to find files that match the pattern and to read multiple files at a time
    - Read Matches reads the content of each match file
- Link_data = (files | “Extract Links” >> …) then uses the function ExtractLinksFn from before to get the outgoing links and create a pair of page names and outgoing links
- We then want to process the outgoing links by using
    - Compute outgoing links to apply ComputeLinkStatistics to count links for each page
    - Combine the outgoing into a list
    - Sort and get the top 5 pages for the most outgoing links
- Incoming links processes it by:
    - incoming_counts to cont how many times each page is linked to
    - Flatten links to turn them into pairs of (links,1) to make it easier to count
    - Count incoming links to college the results into a list
    - Sort top 5 incoming links for which page they come from
- We want to write this to the bucket from beam.io because we can not print it
- Get the runtime by subtracting the end time by start time

Output

Top Outgoing Links
[('1303', 249), ('1468', 249), ('1524', 249), ('1732', 249), ('1959', 249)]

https://storage.googleapis.com/jeremybui_ps2/top_outgoing-00000-of-00001

```
[('1303', 249), ('1468', 249), ('1524', 249), ('1732', 249), ('1959', 249)]
```

Top Incoming Links
[('5984', 188), ('1912', 166), ('5789', 163), ('7885', 162), ('7231', 162)]

https://storage.googleapis.com/jeremybui_ps2/top_incoming-00000-of-00001

```
[('5984', 188), ('1912', 166), ('5789', 163), ('7885', 162), ('7231', 162)]
```

Runtime Locally (DirectRunner)

```
/mnt/c/Users/jerem/dev/github/ds561/07-apache-beam-and-dataflow-bui-jeremy$ python3 main.py
Running pipeline...
WARNING:apache_beam.io.filebasedsink:Deleting 1 existing files in target path matching: -*-of-%(num_sh
ards)05d
WARNING:apache_beam.io.filebasedsink:Deleting 1 existing files in target path matching: -*-of-%(num_sh
ards)05d
Pipeline runtime: 1971.60 seconds
```

1971.60 seconds / 32.86 minutes

Runtime Dataflow (DataflowRunner)

```
/mnt/c/Users/jerem/dev/github/ds561/07-apache-beam-and-dataflow-bui-jeremy$ python3 main.py
Running pipeline...
WARNING:apache_beam.options.pipeline_options:Bucket specified in temp_location has soft-delete policy
enabled. To avoid being billed for unnecessary storage costs, turn off the soft delete feature on buck
ets that your Dataflow jobs use for temporary and staging storage. For more information, see https://c
loud.google.com/storage/docs/use-soft-delete#remove-soft-delete-policy.
WARNING:apache_beam.options.pipeline_options:Bucket specified in staging_location has soft-delete poli
cy enabled. To avoid being billed for unnecessary storage costs, turn off the soft delete feature on b
uckets that your Dataflow jobs use for temporary and staging storage. For more information, see https:
//cloud.google.com/storage/docs/use-soft-delete#remove-soft-delete-policy.
WARNING:apache_beam.options.pipeline_options:Bucket specified in temp_location has soft-delete policy
enabled. To avoid being billed for unnecessary storage costs, turn off the soft delete feature on buck
ets that your Dataflow jobs use for temporary and staging storage. For more information, see https://c
loud.google.com/storage/docs/use-soft-delete#remove-soft-delete-policy.
WARNING:apache_beam.options.pipeline_options:Bucket specified in staging_location has soft-delete poli
cy enabled. To avoid being billed for unnecessary storage costs, turn off the soft delete feature on b
uckets that your Dataflow jobs use for temporary and staging storage. For more information, see https:
//cloud.google.com/storage/docs/use-soft-delete#remove-soft-delete-policy.
.Pipeline runtime: 1168.96 seconds
```

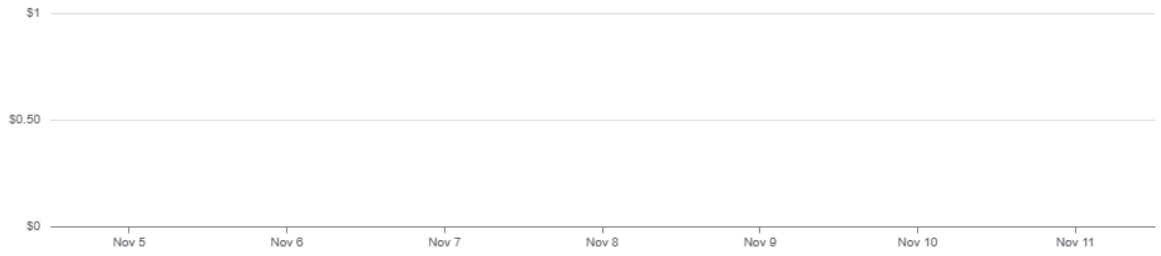
1168.96 seconds / 19.4826667 minutes

The DataflowRunner was faster due to distributing tasks across multiplied optimized cloud workers and benefit from the parallel processing. The DirectRunner was running locally and limited to my laptop's slower resources, network latency, nad less optimized sequential processing.

Your total cost (November 5 – 11, 2024)

Last 7 days | Current month

Cost		Credits used		Total cost	Forecasted total cost
$5.70	—	$5.70	=	$0.00	$0.00
		View details			0% vs. prev. 7d



→ View details on Reports