DataEng: Data Maintenance In-class Assignment

Chase Verbout

This week you will construct a data archiver that compresses, encrypts and stores pipelined data into a low-cost, high-capacity GCP Storage Bucket.

Submit: Make a copy of this document and use it to record your responses and results (use colored highlighting when recording your responses/results). Store a PDF copy of the document in your git repository along with your code before submitting for this week.

Develop a new python PubSub subscriber similar to the subscribers that you have created multiple times for this class. This new subscriber (archive.py) will receive data from a PubSub topic, compress the data, encrypt the data and store the resulting data into a GCP Storage Bucket.

A. [MUST] Discussion Questions

When archiving data for a data pipeline we could (a) compress, (b) encrypt and/or (c) reduce the data. Here, “reducing the data” refers to the process of interpolating or aggregating detailed data, such as 5 second breadcrumbs for all buses on all trips, into coarser data. For example, we could aggregate 5-second breadcrumbs into 30-second breadcrumbs.

Under what circumstances might each of these transformations (compress, encrypt, reduce) be desirable for data archival?

Would it make sense to combine these transformations?

Compress: This is good for making the data smaller in storage- think zip files. We might need this if storage is expensive or limited as a resource.

Encrypt: This is a means for security and privacy. If you have critical information that needs to be shielded from visibility, encryption obscures it.

Reduce: Reduction can be useful for both increasing readability in the data or privacy in removal of sensitive information. Also makes data smaller for storing.

Combined: They can certainly be used in tandem however the order is very important. Encryption should occur after compression and not before. Compression is reliant on the patterns in the data and encryption removes these patterns.

B. [MUST] Create Test Pipeline

Create a new PubSub topic called “archivetest” or something similar. Create a new subscriber program (call it archiver.py) that subscribes to the topic, receives the data and (for now) discards it.

To produce test data, copy/reuse the publisher program used for your class project, and alter it to publish to the new archivetest topic. Run this test publisher manually to gather data (1 day and 100 vehicles) from busdata.cs.pdx.edu and test the archivetest topic and your archiver.py program.

As always, you can/should test your code with smaller data sets first. Try it with just one bus or one trip, and then when everything is working, run it with 100 vehicles.

archiver.py | pub.py | sub.py | topic_clean.py

pub.py > publish_message

```python
from google.cloud import pubsub_v1
import json
import requests
import pandas as pd

# TODO(developer)
project_id = "data-eng-420421"
topic_id = "archive-test"

publisher = pubsub_v1.PublisherClient()
# The `topic_path` method creates a fully qualified identifier
# in the form `projects/{project_id}/topics/{topic_id}`
topic_path = publisher.topic_path(project_id, topic_id)



def publish_message(data):
    data = json.dumps(data).encode('utf-8')
    future = publisher.publish(topic_path, data)
    return future.result()

def get_vehicle_ids():
    doc_key = "10VKMye65LhbEgMLld5Ol3lOocWUwCaEgnPVgFQf9em0"
    url = f"https://docs.google.com/spreadsheets/d/{doc_key}/export?format=csv"
    response = requests.get(url)
    csv_data = response.content
    with open("vehicle_ids_sheet.csv", "wb") as file:
        file.write(csv_data)
    vehicle_ids = pd.read_csv("vehicle_ids_sheet.csv")['Doodle'].tolist()
    return vehicle_ids

def gather_and_publish_data():
    vehicle_ids = get_vehicle_ids()
```

```python

def gather_and_publish_data():
    vehicle_ids = get_vehicle_ids()
    for vehicle_id in vehicle_ids:
        api_url = f"https://busdata.cs.pdx.edu/api/getBreadCrumbs?vehicle_id={vehicle_id}"
        response = requests.get(api_url)
        if response.status_code == 200:
            data = response.json()
            publish_message(json.dumps(data))
        else:
            print(f"Failed to fetch data for vehicle ID {vehicle_id}")


gather_and_publish_data()

print(f"Published messages to {topic_path}.")
```

archiver.py | pub.py | sub.py | topic_clean.py

archiver.py > subscription_path

```python
import json
from google.cloud import pubsub_v1
from google.cloud import storage
from datetime import datetime
from threading import Timer

project_id = "data-eng-420421"
subscription_id = "archive-test-sub"
bucket_name = "inclass---8"
buffer_time = 60
print_status_interval = 300
buffer = []
subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_id)
storage_client = storage.Client()

def upload_to_bucket(data, bucket_name):
    timestamp = datetime.utcnow().strftime('%Y%m%d%H%M%S')
    file_name = f"breadcrumb_{timestamp}.json"

    bucket = storage_client.get_bucket(bucket_name)

    blob = bucket.blob(file_name)
    blob.upload_from_string(json.dumps(data), content_type="application/json")

    print(f"Uploaded {file_name} to {bucket_name}")

def save_buffer():
    global buffer
    if buffer:
        upload_to_bucket(buffer, bucket_name)
        buffer = []

```

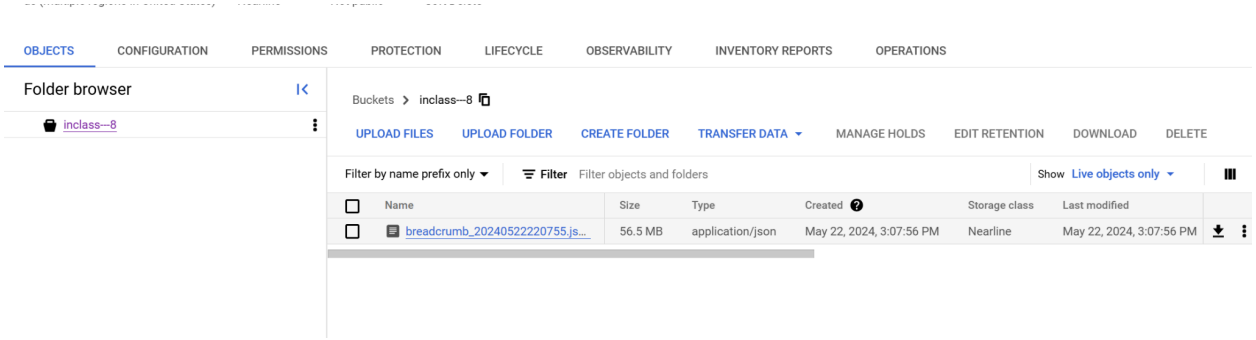
Ln 14, Col 78 Spaces: 4 UTF-8 LF

```
28  def save_buffer():
29      global buffer
30      if buffer:
31          upload_to_bucket(buffer, bucket_name)
32          buffer = []
33
34      Timer(buffer_time, save_buffer).start()
35
36  def print_status():
37      print(f"Buffer contains {len(buffer)} messages.")
38      Timer(print_status_interval, print_status).start()
39
40  def callback(message):
41      global buffer
42      buffer.append(json.loads(message.data.decode('utf-8')))
43      message.ack()
44
45  save_buffer()
46
47  print_status()
48
49  streaming_pull_future = subscriber.subscribe(subscription_path, callback=callback)
50  print(f"Listening for messages on {subscription_path}...")
51
52  with subscriber:
53      try:
54          streaming_pull_future.result()
55      except TimeoutError:
56          streaming_pull_future.cancel()
57
```

C. [MUST] Store Data to GCP Storage Bucket

Modify archiver.py to store all received data to a GCP Storage Bucket. You will need to create and configure a Storage Bucket for this purpose. We recommend using the Nearline Storage class for this assignment though you are free to choose any of the offered classes of service. Be sure to remove the bucket at the end of the week to reduce GCP credit usage.

How much bucket space (in KiBs) does it take to store 1 day of breadcrumbs for 100 vehicles?



56.5MB = 55175.78125 KiB

D. [SHOULD] Compress

Modify archiver.py to compress the data before it stores the data to the storage bucket. Use zlib compression which is provided by default by python. How large is the archived data compared to the original?

How much bucket space (in KiBs) does it take to store the compressed data?

E. [SHOULD] Encrypt

Modify archiver.py to encrypt the data prior to writing it to the Storage Bucket. Your archive.py program should encrypt after compressing the data. Use RSA encryption as described here: link There is no need to manage your private encryption keys securely for this assignment, and you may keep your private key in a file or within your python code.

Be sure to test your archiver by decrypting and decompressing the data stored in the Storage Bucket. We suggest that you create a separate python program for this purpose.

How much bucket space (in KiBs) does it take to store the encrypted, compressed data?

F. [ASPIRE] Add Archiving to your class project

Add an archiver to your class project's pipeline(s). To receive extra credit, mention your archiver when submitting the next part of your project. You should only need one archiver for the entire project, so coordinate with your teammates if you choose to take this step. For the class project, we recommend storing to a Google Storage Bucket and compressing. Encryption is OK too but not necessary.