DataEng: Data Maintenance In-class Assignment

This week you will gain hands-on experience with Data Maintenance by constructing an automated 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 results. Store a PDF copy of the document in your git repository along with any needed code before submitting for this week.

Your job is to develop a new, separate python Kafka consumer similar to the consumers that you have created multiple times for this class. This new consumer should be called archive.py and should receive all data from a test Kafka topic, compress the data, encrypt the data (optional) and store the compressed data in a GCP Storage Bucket.

Note that each member of your team should build their own archive mechanism. As always, it is OK to help one another, but each person should develop their own python program for archiving.

Discussion Question for Your Entire Group (do this first)

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 transforming detailed data, such as 5 second breadcrumbs for all buses on all trips, into coarser data that contains only a subset of the original data such as only some buses, some trips or possibly fewer breadcrumbs per trip.

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

My Response: The transformation such as compression can be useful in cases where limited storage resources need to be implemented in order to achieve cost effective and efficient data storage. Applications that contain customer sensitive data such as bank details, personal details need encryption for safe storage and archival. Reduction can be useful for storing relevant application data and discarding redundant irrelevant data. Reduction can also help in performing data analysis on the relevant data.

Kavitha Jajula Response: I think Employee datasets are good for transformations. Since it is huge data it can be compressed, and it has personal sensitive information of each employee so it needs to be encrypted. Also reducing the data can be done based on the employee information present in the system. For example, if there is a date of birth of an employee in the database, the age column can be removed which is not necessary.

Shatabdi Pal Response: For the TriMet breadcrumb dataset, we can do compression for archival. The data sets with patient information contain sensitive information. In that case data archival needs encryption to avoid privacy policy related issues. We removed unnecessary columns from stop event data before loading them into the database. We can follow the same process to reduce the dataset for archival.

Shreshtha Siddhi Response: I can think of following circumstances for each of the three transformations:
1.Compress: Netflix data such as videos can be a good example which can be used with Data Compression. Compression will help in faster streaming for the audience, by reducing the size of the videos.
2.Encrypt: Encryption is used to safeguard the information being transmitted whenever someone uses an ATM or makes an online purchase using a smartphone. Also, Businesses are depending more and more on encryption to shield sensitive data and applications from data breach.
3.Reduce: Data reduction process is utilized in wearable (wireless) devices for health monitoring and diagnosis applications. For e.g., detecting neurological disorders.

Part 1: Data Archival to Cold Storage

A. Create test topic

Create new Kafka producer and consumer programs as you did with the Data Transport in-class activity. Create a new Kafka topic that is separate from the topic(s) used for your project. Call it “archivetest” or something similar. As with the Data Transport activity you should initially have your new producer produce test data and have a single consumer/archiver (call it “archiver.py” or something similar) that consumes any/all data sent to the Kafka topic and simply prints out the data consumed.

B. Create separate consumer groups

Create two separate Consumer Groups for your new Kafka topic. Run a separate consumer for each group and verify that each consumer consumes all of the data sent by the producer. The first of these two consumers will simulate your primary pipeline’s consumer. In the next step you will modify the second consumer to archive the data.

Answer: I created two separate getting_started.ini files each with different group.id for configuring two consumers in different groups. However, I was not able to download these files from the GCP instance so I have not committed them to the git repository.

C. Archive the data

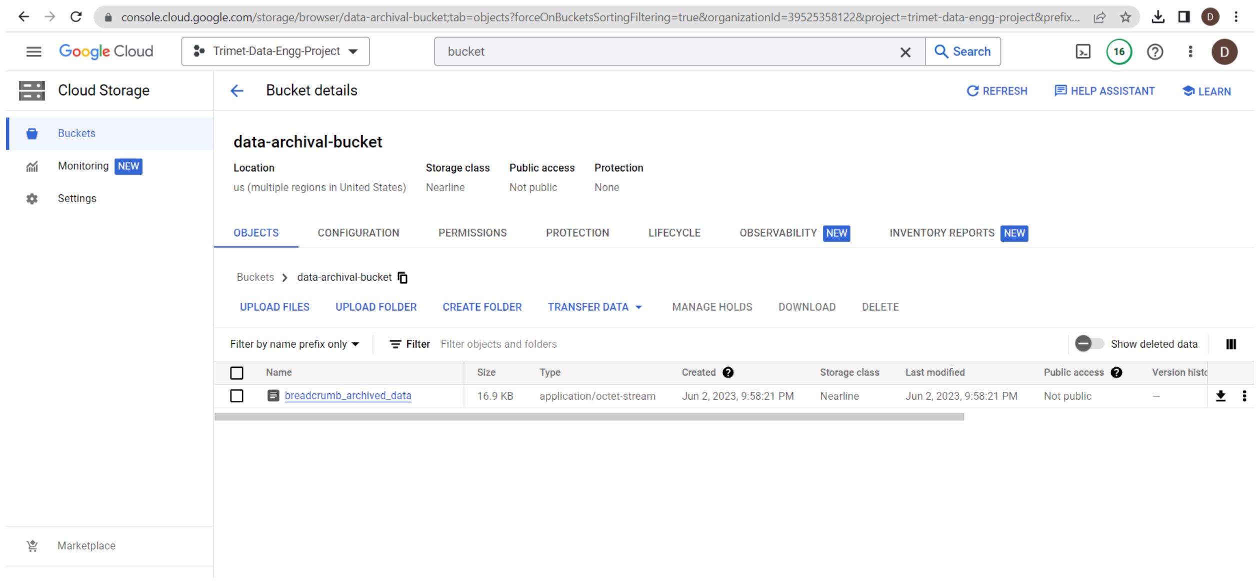
Modify the second of the two consumers (rename it “archiver.py” or something similar) to store all received data into a file.

Next, modify the archiver to store all received data to a GCP Storage Bucket. You will need to create and configure a Storage Bucket for this purpose. You are free to choose any of the available storage classes. We recommend using the Nearline Storage class for this assignment. Be sure to remove the bucket at the end of the week to reduce GCP credit usage.

D. Compress (Optional)

Modify your archiver program 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?
Answer: The archived data is 94% smaller in size than the original data.

Below is the screenshot of the data stored in the GCP bucket:



data-archival-bucket

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Nearline	Not public	None

Name	Size	Type	Created	Storage class	Last modified	Public access	Version histo
breadcrumb_archived_data	16.9 KB	application/octet-stream	Jun 2, 2023, 9:58:21 PM	Nearline	Jun 2, 2023, 9:58:21 PM	Not public	–

E. Encrypt (Optional)

Modify your archiver 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 large is the archived data?

F. Add Archiving to your class project (Optional)

Add an archiver to your class project’s pipeline(s). Mention this in your final project presentation video. 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, it is not necessary to securely manage your RSA private encryption key, and it is OK to keep it in a file or in your python source code.