DataEng: Data Maintenance In-class Assignment

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
Purpose: Reduce the size of the data to save storage space and improve transfer speed.
Desirable When:
Storage Efficiency: When storage resources are limited or expensive, compressing data can significantly reduce the amount of storage needed.
Transmission Speed: When data needs to be transmitted over a network, compression can speed up the process by reducing the data size.
Cost Savings: Lower storage costs and faster data transfer can result in reduced overall costs.
Encrypt:
Purpose: Protect the data from unauthorized access and ensure confidentiality.
Desirable When:
Security and Privacy: When the data contains sensitive information such as personal, financial, or proprietary data, encryption is essential to protect it from unauthorized access.
Compliance: When regulations and standards (e.g., GDPR, HIPAA) require data to be encrypted to ensure privacy and security.
Risk Management: When mitigating risks associated with data breaches and cyber attacks is a priority.

Reduce
Purpose: Summarize or condense the data by removing unnecessary or redundant information.
Desirable When:
Relevance: When only a subset of the data is needed for future reference, reducing the data helps to keep only relevant information.
Efficiency: When working with large datasets, reducing the data can make analysis more efficient and manageable.
Cost: Like compression, reducing data can lower storage costs by keeping only essential information.
Combining Transformations
Combining these transformations can be both practical and beneficial:
Compress + Encrypt: Compress the data first to reduce its size and then encrypt it to ensure security. This order is crucial because encrypted data typically does not compress well due to its randomized nature.
Encrypt + Reduce: Encrypting sensitive data can be done before reducing it to ensure that even the reduced dataset remains secure. However, this approach may depend on the specific use case and whether the reduced data needs to retain its original security context.
Reduce + Compress: Reduce the data first to eliminate unnecessary parts, then compress the remaining data to save space. This combination maximizes storage efficiency by first minimizing the data volume and then compressing it.

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

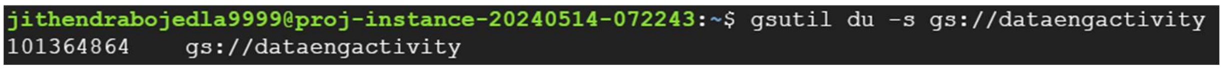
```
import os
from google.cloud import pubsub_v1
from google.cloud import storage
from datetime import datetime

# Project ID and Subscription details
project_id = "scientific-pad-420219"
subscription_name = "archeivetest-sub"
bucket_name = "dataengactivity"

# Initialize a Subscriber client
subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_name)
```

```
# Initialize a Storage client
storage_client = storage.Client()
bucket = storage_client.bucket(bucket_name)

def callback(message):
    print(f"Received message: {message.data}")
    # Create a unique filename for each message based on the current timestamp
    timestamp = datetime.utcnow().strftime('%Y%m%d%H%M%S%f')
    filename = f"breadcrumb_{timestamp}.txt"
    blob = bucket.blob(filename)
    blob.upload_from_string(message.data.decode('utf-8'))
    print(f"Stored message to {filename} in bucket {bucket_name}")
    message.ack()  # Acknowledge the message

# Create the subscription if it doesn't exist
def create_subscription():
    topic_path = subscriber.topic_path(project_id, "archeivetest")
    try:
        subscriber.create_subscription(name=subscription_path, topic=topic_path)
        print(f"Subscription {subscription_name} created.")
    except Exception as e:
        print(f"Subscription {subscription_name} already exists or failed to
create: {e}")

create_subscription()

# Subscribe to the topic
streaming_pull_future = subscriber.subscribe(subscription_path,
callback=callback)
print(f"Listening for messages on {subscription_path}...")

# Block the main thread while waiting for messages
try:
    streaming_pull_future.result()
except KeyboardInterrupt:
    streaming_pull_future.cancel()
```

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?

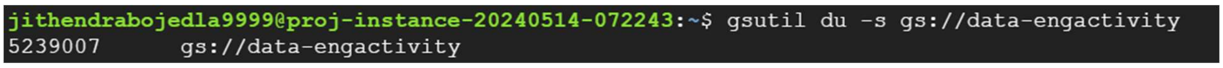


```
jithendrabojedla9999@proj-instance-20240514-072243:~$ gsutil du -s gs://dataengactivity
101364864    gs://dataengactivity
```

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?



```
jithendrabojedla9999@proj-instance-20240514-072243:~$ gsutil du -s gs://data-engactivity
5239007      gs://data-engactivity
```

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.

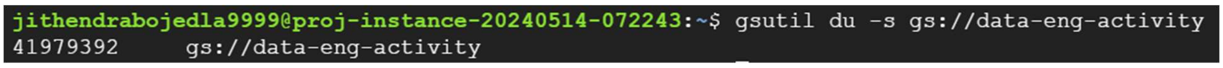
new_archiver.py

```python
import os
import zlib
from google.cloud import pubsub_v1
from google.cloud import storage
from datetime import datetime
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP

# Project ID and Subscription details
project_id = "scientific-pad-420219"
subscription_name = "archeivetest-sub"
bucket_name = "data-eng-activity"

# Initialize a Subscriber client
subscriber = pubsub_v1.SubscriberClient()
subscription_path = subscriber.subscription_path(project_id, subscription_name)
```

```python
# Initialize a Storage client
storage_client = storage.Client()
bucket = storage_client.bucket(bucket_name)

# RSA key generation and saving the private key
key = RSA.generate(2048)
private_key = key.export_key()
with open("private.pem", "wb") as f:
    f.write(private_key)

# Public key for encryption
public_key = key.publickey().export_key()
public_key = RSA.import_key(public_key)
cipher_rsa = PKCS1_OAEP.new(public_key)

def callback(message):
    print(f"Received message: {message.data}")
    # Create a unique filename for each message based on the current timestamp
    timestamp = datetime.utcnow().strftime('%Y%m%d%H%M%S%f')
    filename = f"breadcrumb_{timestamp}.txt"

    # Compress the message data using zlib
    compressed_data = zlib.compress(message.data)

    # Encrypt the compressed data using RSA
    encrypted_data = cipher_rsa.encrypt(compressed_data)
    encrypted_filename = f"{filename}.zlib.enc"

    # Upload the encrypted data to the bucket
    blob = bucket.blob(encrypted_filename)
    blob.upload_from_string(encrypted_data)

    print(f"Stored encrypted and compressed message to {encrypted_filename} in
bucket {bucket_name}")
    message.ack()  # Acknowledge the message

# Create the subscription if it doesn't exist
def create_subscription():
    topic_path = subscriber.topic_path(project_id, "archeivetest")
    try:
        subscriber.create_subscription(name=subscription_path, topic=topic_path)
        print(f"Subscription {subscription_name} created.")
    except Exception as e:
        print(f"Subscription {subscription_name} already exists or failed to
create: {e}")
```

```
create_subscription()

# Subscribe to the topic
streaming_pull_future = subscriber.subscribe(subscription_path,
callback=callback)
print(f"Listening for messages on {subscription_path}...")

# Block the main thread while waiting for messages
try:
    streaming_pull_future.result()
except KeyboardInterrupt:
    streaming_pull_future.cancel()
```

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



```
jithendrabojedla9999@proj-instance-20240514-072243:~$ gsutil du -s gs://data-eng-activity
41979392     gs://data-eng-activity
```

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.