Prueba Teórica Proyecto Falabella Logs

Se tiene un AVRO de **60 millones de registros**, se requiere realizar la **encriptación de 5 de las 10 columnas** (todos datos tipo **string de no más de 32 bytes**). Qué componentes de GCP y lenguaje (o lenguajes) de programación utilizaría para realizar esta transformación de la forma más eficiente y con el mejor tiempo de respuesta? Explique por qué

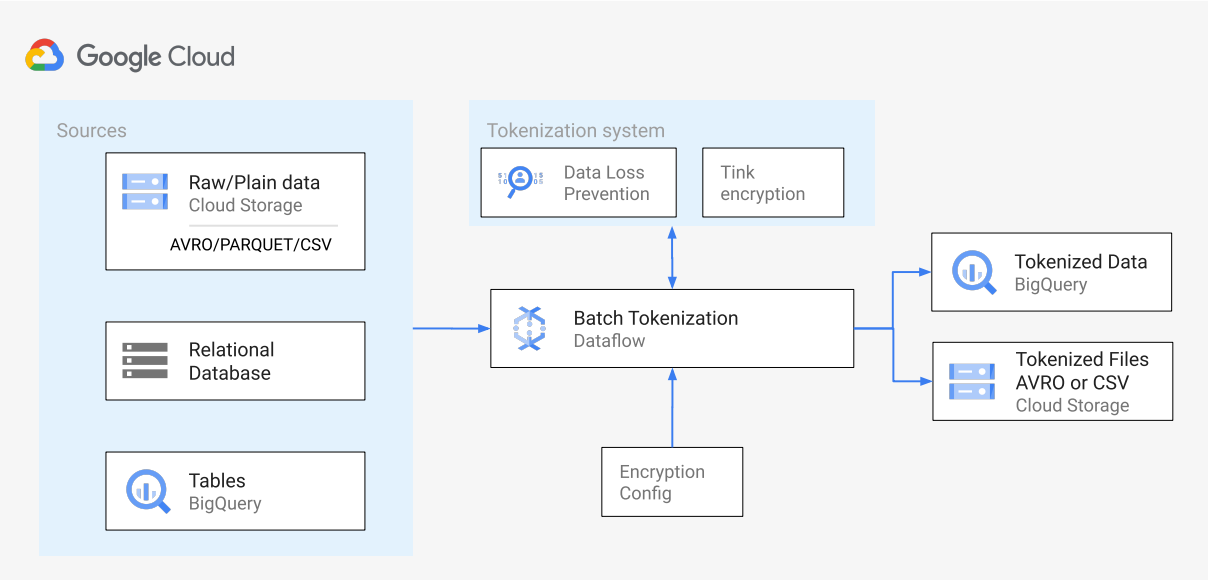
Se proponen tres soluciones de GCP para cumplir con la necesidad descrita, estas se presentan a continuación, dado que algunas de las herramientas a utilizar son comunes entre las diferentes alternativas estas se describen previamente en el siguiente bloque:

Herramientas

- Cloud KMS: Servicio de GCP para administrar de forma segura claves de encriptación en google cloud, permite almacenar y proveer de manera segura en los diferentes componentes y servicios de un workflow que requieran dichas claves. Se utilizaría la tenica de envelope encyption para agregar seguridad a la clave de encriptación de los datos pues esta será reversible con el fin de que luego se peuda acceder nuevamente a los datos.
- Dataflow: Servicio serverless autoescalable (en caso tal que con este pipeline se requiera procesar nuevos archivos con diferentes cantidades de registro y complejidad, el auto escalamiento ahorraría tener que ajustarse a los requerimientos de cada archivo a procesar). Requiere crear una cuenta de servicio para la ejecución de los pipelines. En este caso se haría uso del SDK de Python de Apache Beam.
- Cloud Storage: Se necesitaría al menos un bucket en el que almacenar el archivo avro a ser procesado (zona raw o landing) y un bucket de salida en que se almacene el archivo AVRO procesado con las columnas encriptadas.
- Tink: https://github.com/google/tink Librería de codigo abierto de Google, que provee herramientas para encriptar y desencriptar cuaqluier tipo de datos. Desarrollada luego de que el proyecto de Google https://github.com/google/wycheproof detectara varios fallos de seguridad en otras librerías ampliamente usadas por la comunidad, y su objetivo es solventar estas brechas de seguridad y proveer alto rendimiento de la nube. Las funciones de encriptación de esta librería están desarrolladas en **C++,** lo que permite obtener un alto rendimiento en terminos de uso de memoría y velocidad de procesamiento, además cuenta con APIs para diferentes lenguajes de progrmación como Python y Java, que simplemente funcionan como wrappers de las funciones de C++ por lo que se decide utilizar la librería de Python, dada la facilidad de uso sin perder rendimiento. Tink es compatible con Cloud KMS, por lo que puede acceder a la clave de encriptación para utilizar envelope encryption. También se utilizará la librería de JAVA en la solución propuesta 1, que corresponde a un pipeline de Google disponible publicamente en GitHub.
- Bigquery: Una alternativa es cargar el archivo AVRO en bq y aprovechar su integración con clud KMS para generar una tabla con columnas encriptadas

Solución 1: Implementar pipeline de dataflow (Java) disponible en el github de GoogleCloudPlatform

A continuación se presenta la arquitectura de la solución implementando un pipeline disponible publicamente en el repositorio de GitHub para la encriptación de columnas de archivos AVRO en GCP



Procedimiento

1. Configurar cloud storage: crear un bucket de entrada en que cargar el archivo AVRO que se va a procesar, y un bucket de salida en que se pueda almacenar el archivo AVRO una vez sea procesado en Dataflow. Esto se realizaría por medio de Terraform o con la utilidad de shell gcloud mb.
2. Crear llaves de encriptación en cloud KMS: Los datos serán encriptados usando una Data encryption key (DEK), luego se utiliza el metodo envelope encryption, así, esta llave de enciprtación es encriptada y almacenada en el cloud KMS, para esto se debe configurar una Key encryption key (KEK), haciendo que la clave de encriptación de los datos tenga una capa de seguridad adicional. Esto se puede realizar con la utilidad de shell gcloud kms.
3. Procesamiento del archivo AVRO en Dataflow: Para esto se cuenta con una interfaz para la ejecución del pipeline disponible en https://github.com/GoogleCloudPlatform/auto-data-tokenize, el cual permite seleccionar columnas de interes en un archivo AVRO que se encuentre en un bucket, para ser encriptadas y luego disponer el archivo de salida en otro bucket de salida. En este Pipeline se ejecutan (tomado de la docuemtnación) los siguientes procedimientos:

> 1. Unwrap the data encryption key using Cloud KMS
> 2. Un-nest each record into a flat record.
> 3. Tokenize required values using deterministic AEAD.
> 4. Re-nest the flat record into an Avro record.
> 5. Write an Avro file with encrypted fields.

3. Este Pipeline se ejecuta de la siguiente manera:

```
# Clonar el repo
git clone https://github.com/GoogleCloudPlatform/auto-data-tokenize.git
cd auto-data-tokenize/

# Definir las variables de entorno del proyecto

export PROJECT_ID="<your-project-id>"
export REGION_ID="<compute-engine-region>"
export TEMP_GCS_BUCKET="<name-of-the-bucket>"
export KMS_KEYRING_ID="<key-ring-name>"
export KMS_KEY_ID="<key-id>"
export WRAPPED_KEY_FILE="<path-to-the-data-encryption-key-file>" # generado con tink

# Crear llaves de encriptación
gcloud kms keyrings create --project ${PROJECT_ID} --location ${REGION_ID} ${KMS_KEYRING_ID}
```

```
gcloud kms keys create --project ${PROJECT_ID} --keyring=${KMS_KEYRING_ID} --location=${REGION_ID} --purpose="encryption" ${KMS_KEY_ID}


# Descargar y configurar Tinkey - Crear llave de enciprtación para encriptar la llave de encriptación de los datos

mkdir tinkey/
tar zxf tinkey-<version>.tar.gz -C tinkey/
export TINKEY="${PWD}/tinkey/tinkey"
alias tinkey="${TINKEY}"

tinkey create-keyset \
--master-key-uri="${MAIN_KMS_KEY_URI}" \
--key-template=AES256_SIV \
--out="${WRAPPED_KEY_FILE}" \
--out-format=json


# Ejecutar pipeline para las columnas deseadas
# Como tokenizeColumns se indican cada una de las 5 columnas a encriptar

tokenize_pipeline --project="${PROJECT_ID}" \
--region="${REGION_ID}" \
--runner="DataflowRunner" \
--tempLocation="gs://${TEMP_GCS_BUCKET}/bqtemp" \
--workerMachineType="n1-standard-1" \ # Seleccionar el tipo de maquina en que se porcesara el archivo
--schema="$(<dlp_report/schema.json)" \ # esquema del archivo AVRO
--tinkEncryptionKeySetJson="$(<${WRAPPED_KEY_FILE})" \ # llave creada con tink
--mainKmsKeyUri="${MAIN_KMS_KEY_URI}" \
--sourceType="AVRO" \
--inputPattern="gs://${TEMP_GCS_BUCKET}/userdata.avro" \ #bucket de entrada
--outputDirectory="gs://${TEMP_GCS_BUCKET}/encrypted/" \ #bucket de salida
--tokenizeColumns="$.kylosample.cc" \
--tokenizeColumns="$.kylosample.email"
```

Solución 2: Implementar pipeline de dataflow (Python) con estrategia similar a solución 1

Procedimiento

1. Configurar cloud storage: crear un bucket de entrada en que cargar el archivo AVRO que se va a procesar, y un bucket de salida en que se pueda almacenar el archivo AVRO una vez sea procesado en Dataflow. Esto se realizaría por medio de Terraform o con la utilidad de shell gcloud mb.
2. Crear llaves de encriptación en cloud KMS: Los datos serán encriptados usando una Data encryption key (DEK), luego se utiliza el metodo envelope encryption, así, esta llave de enciprtación es encriptada y almacenada en el cloud KMS, para esto se debe configurar una Key encryption key (KEK), haciendo que la clave de encriptación de los datos tenga una capa de seguridad adicional. Esto se puede realizar con la utilidad de shell gcloud kms.
3. Procesamiento del archivo AVRO en Dataflow: Para esto se configura como un parametro de entrada del pipeline de dataflow, el nombre del bucket y del archivo a ser procesado, además los nombres de las columnas que deben ser encriptadas. Este archivo es cargado en la instancia de ejecución de Dataflow y se utiliza el modulo de AVRO de Apache Beam (Python) para la lectura y escritura de archivos AVRO (Apacha Beam doc), se carga la metadata del archivo para obtener el esquema del archivo (este será modificado a los tipos de salida de las columnas encriptadas). Luego se itera por cada una de las columnas para generar llaves de encriptación y aplicar el metodo de encriptación sobre cada una de las columnas indicadas. El codigo que se utilizaría (tomado de la docuemtnación) sería similar al siguiente (se implementa el acceso al clud KMS para utilizar la encriptación de tipo envelope):

```
import tink
from tink import aead
from tink.integration import gcpkms
from google.cloud import storage


storage_client = storage.Client()
bucket = storage_client.get_bucket(INPUT_BUCKET)
blob = bucket.get_blob(path_file)
blob.download_to_filename(temp_file_name)
```

```
key_uri = 'gcp-kms://projects/tink-examples/locations/global/keyRings/foo/cryptoKeys/bar'
gcp_credentials = 'credentials.json'

""
Lectura del archivo AVRO y definición del esquema indicando las columnas a ecnritpar como bytes

Se utilizaria la clase de lectura de archivos AVRO de apache beam

apache_beam.io.avroio.ReadFromAvro(file_pattern=None, min_bundle_size=0, validate=True)[source]
""

# Read the GCP credentials and setup client
try:
  gcp_client = gcpkms.GcpKmsClient(key_uri, gcp_credentials)
  gcp_aead = gcp_client.get_aead(key_uri)
except tink.TinkError as e:
  logging.error('Error initializing GCP client: %s', e)
  return 1

# Create envelope AEAD primitive using AES256 GCM for encrypting the data
try:
  key_template = aead.aead_key_templates.AES256_GCM
  env_aead = aead.KmsEnvelopeAead(key_template, gcp_aead)
except tink.TinkError as e:
  logging.error('Error creating primitive: %s', e)
  return 1

# Use env_aead to encrypt data
# Esto se aplicaria sobre cada una de las columnas a encriptar
ciphertext = env_aead.encrypt(plaintext, associated_data)


""
Escritura del archivo AVRO encriptado

Se utilizaria la clase de escritura de archivos AVRO de apache beam

apache_beam.io.avroio.WriteToAvro(file_path_prefix, schema, codec='deflate', file_name_suffix='', num_shards=0, shard_name_template=None, m

""


output_bucket = storage_client.bucket(OUTPUT_BUCKET)
output_blob = output_bucket.blob(output_filename)
output_blob.upload_from_filename(output_filename)
```

4. El AVRO escriptado se escribiria en el temp de la instancia configurada para la ejecución de DataFlow y se utilizaría la librería de cloud sorage apr escribir en el bucket de salida

Solución 3: Encriptar las columnas requeridas en bigquery con Cloud KMS y exportar AVRO a bucket de salida

A esta solución podrían agregarse triggers para automatizar procedimiento para cada archivo AVRO que fuese cargado en el bucket según necesidades, pero en este caso se describe cómo funcionaría el procedimiento para un solo archivo:

Procedimiento

1. Cargar archivo AVRO a bigquery desde el bucket de entrada usando en la consola:

```
bq load --source_format=AVRO nombredataset.nombretabla "gs://bucket-entrada/carpeta/*.avro"
```

2. Crear una segunda tabla en que se insertara los datos de la primera con las columnas encritpadas, para esto se debe generar el siguiente esquema que permita la encriptación:

```
CREATE TABLE IF NOT EXISTS nombredataset.nombretabla_con_encriptacion (
columna STRING,
columna_encriptada BYTES);
#agregar todas las columnas necesarias - para este caso 5 columnas encriptadas
```

3. Crear una llave de ecnriptación en bigquery para las columnas a encriptar

```
$ bq --project_id=id_proyecto query --use_legacy_sql=false "SELECT KEYS.NEW_KEYSET('AEAD_AES_GCM_256') AS raw_keyset"
```

4. Generar la wrapped key que asegurara la clave de encriptación de los datos y será almacenada en cloud KMS

```
echo "CPPpiNMFEmQKWAowdHlwZS5nb29…" |base64 --decode > /tmp/decoded_key
gcloud kms encrypt --plaintext-file=/tmp/decoded_key --key=projects/PROJECT_ID/locations/LOCATION_NAME/keyRings/KEY_RING_ID/cryptoKeys/KEY_
od -An --format=o1 /tmp/bankaccounts_wrapped |tr -d '\n'|tr ' ' '\'
```

5. Encriptar las columnas requeridas en BigQuery

```
DECLARE KMS_RESOURCE_NAME STRING;
DECLARE FIRST_LEVEL_KEYSET BYTES;
SET KMS_RESOURCE_NAME =  "gcp-kms://projects/PROJECT_ID/locations/LOCATION_NAME/keyRings/KEY_RING_ID/cryptoKeys/KEY_ID";
# Set the first level keyset equal to the output from the previous step
SET FIRST_LEVEL_KEYSET = b"\012\044\000\031\261\001\270\114\176\037\055\330.....";
INSERT INTO DATASET_NAME.TABLE_NAME
SELECT "example_plaintext", AEAD.ENCRYPT(KEYS.KEYSET_CHAIN(KMS_RESOURCE_NAME, FIRST_LEVEL_KEYSET), "example_plaintext", "additional_data");
```

6. Exportar archivo AVRO a bucket de salida

```
bq extract --destination_format AVRO nombredataset.nombretabla_encriptada gs://bucket-salida/archivoencriptado*.avro
```