Blockchain Data Aggregator for Marketplace Analytics

Working demo available here - https://youtu.be/mTveDBGiHTE

Requirements

1. Deliverable Summary

- Data extraction, normalization and calculation scripts written in Go
  - Go code implemented in GCP using GCP Functions
- Clickhouse/BQ schema and data insertion scripts
  - DECISION: BigQuery implemented due to easy serverless access from GCP
- Detailed documentation of the process
- Instructions for setting up and running the data pipeline

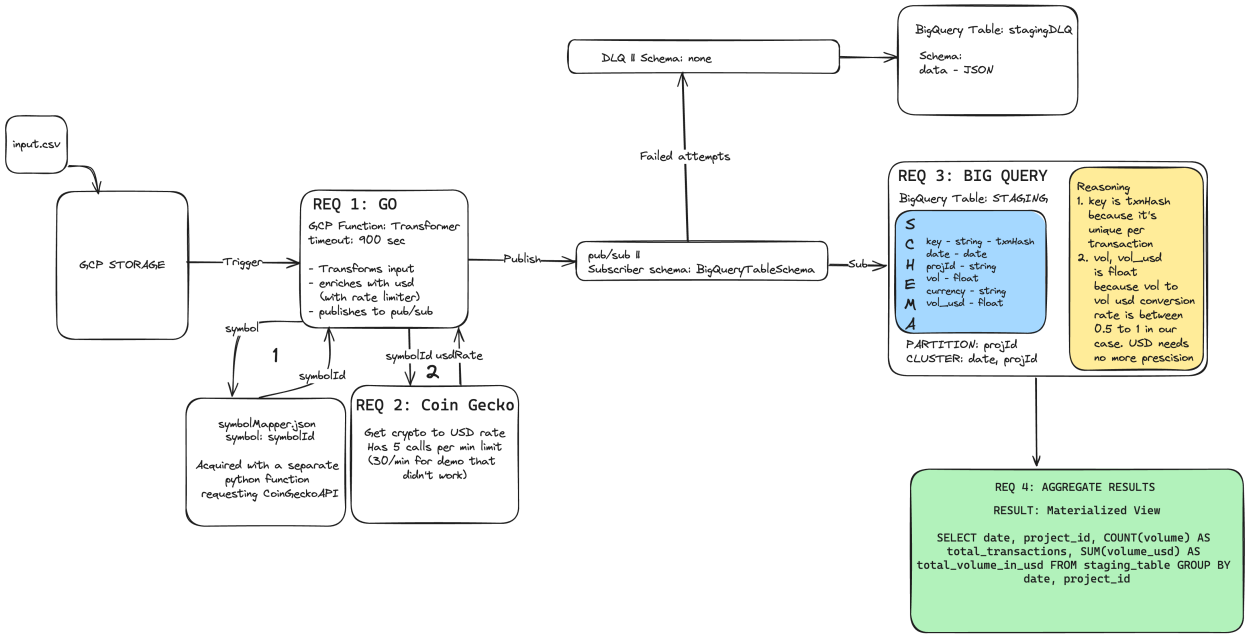
2. Key Go Transformations

1. Collect data from csv - currencySymbol, currencyValueDecimal
2. Flatten Data
3. Request CoinGecko API for USD Rate of cryptocurrency

3. Key BigQuery Transformation

- Aggregate data using Material View

Architecture



Solution: Entire Process

PART 1: GCP Storage Bucket & GCP Functions Trigger

- The GCP Functions is written in Go to meet the test requriement.
- It has a timeout of 900 to allow for rate limiter to take its time to get the usdRate that is needed.

GCP Function - CoinGecko API

- I got all list of coingecko symbol to SymbolID Mapping and stored & transformed it into a json file, using pyspark

```
curl --request GET \
     --url "https://api.coingecko.com/api/v3/coins/list" \
     --header 'accept: application/json' \
     -o coins_list.json
```

```
import json

with open('coins_list.json', 'r') as input_file:
    data = json.load(input_file)
result = {item['symbol']: item['id'] for item in data}
with open('symbol_id_map.json', 'w') as output_file:
    json.dump(result, output_file, indent=4)
```

- This allowed me to request the right data using the SymbolId

```
curl --request GET \
     --url "https://api.coingecko.com/api/v3/simple/price" \
     --header 'accept: application/json' \
     --header 'x-cg-api-key: API_KEY' \
     --get \
     --data-urlencode "ids=bitcoin" \
     --data-urlencode "vs_currencies=usd" \
     --data-urlencode "date=01-01-2024"
```

Hashmap to make API call efficient

- Utilized hashmap to store USDValue of a coin, for particular date.
- Hashmap is referenced before making the API call. Hashmap key would be SymbolId+Date which would give the average rate of any currency in USD.

CoinGecko API Rate Limiter

Issue: API calls overload beyond a point. Initially I thought it was going beyond 30 calls per minute, which wasn't the case. It was much lower at 5 calls per minute due to some account issue.

Solution: Built a rate limiter

```
if callCounter >= 5 {
    time.Sleep(time.Duration(60) * time.Second)
```

```
}
```

PART 2: PUB/SUB

Main Messaging Queue

Here's where the staging table subscribes to, and the Go code publishes the transformed Data.

The schema is the BigQuery table's schema.

Dead Letter Queue

Schema mismatch or any bad data or failed messages go to DLQ. The data of DLQ is stored into BigQuery DLQ table.

Potential Points of Failure that require DLQ

- Transformation error
- Function Timeout
- API request
- Big Query API timeout

Our Actual failure point

volume_usd and volume with extremely high number exceeded "`FLOAT`", and then "`NUMERIC`" and finally exceed "`BIG NUMERIC`" data types, that failed the schema configuration of the subscriber to the pub/sub topic.

```
{"key":"2024-04-02_1609","date":"2024-04-02","project_id":160
9,"volume":2565000000000000000,"currency":"matic-network","vo
lume_usd":13553921700000000300}
```

Data Inconsistency problem

On analysing te data I figured out most data is of range 1,2,3 digits. (since all data is close to 1 usd, I'm going to make all the currencies comparible for this analytics (ignoring precision)

```
In [57]: digit_distribution

Out[57]: digits
         1     707
         2     182
         3      39
         7       6
         18     27
         19     32
         20      7
         dtype: int64
```

- Since digits of “MATIC” have length of 18,19,20 corresponds to 1,2,3, I’m dividing the values by 10^18 - resulting in their values ranging from 0.xx to xx.xx
- For usdc, or usdc.e, some values in decimalValue inconsistently have 7 digits or zero to 1 digits - I’m going to divide any data who’s length is 7 to be divided by 10^7

PART 3: Big Query Staging

- **Aggregation** by - DATE and PROJECT ID
  - sum(transaction)
  - sum(vol_in_usd)
- **Target** Table would look like this

DATE	PROJECT_ID	NUM_TRANSACTIONS	TOTAL_VOLUME_IN_USD
5/10/2024	1001	4	500
5/11/2024	1001	10	1000

Solution 1: Big Query Staging Table Aggregated to Sink [SELECTED]

1. Extract, clean, filter data and drop it into a staging table
2. Aggregate staging table on BigQuery into a materialized view.

~~Solution 2: Aggregate on Go itself with memory~~

- Not viable at the moment since we'd have to make it scalable.
- It's definitely possible with our much smaller dataset for our usecase. But I'll make the assumption that the problem needs to be scalable.

I went with solution 1.

What didn't work

- I was looking to store the data of aggregate directly using MERGE and UPDATE. This isn't something directly possible on BigQuery tables.

PART 4: Big Query Materialized View

- Materialized view of the table that pre-aggregates data for speedy query

```
SELECT
  date,
  project_id,
  COUNT(volume) AS total_transactions,
  SUM(volume_usd) AS total_volume_in_usd
FROM `blockdataproject.blockdata.staging`
GROUP BY date, project_id
```

Steps to run

Working demo available here - https://youtu.be/mTveDBGiHTE

1. In BigQuery page setup the new tables using the schema provided for staging and DLQ in the github folder "`schemas`"
2. Add two pub/sub topics `transformed_data` and `transformed_dlq`. Add two subscribers to write data to BigQuery tables - **`Staging`** and `DLQ`
3. Add new CSV file into `blockdata-input` bucket
4. Finally run the code in

```
select * from blockdataproject.blockdata.staging;
select * from blockdataproject.blockdata.dlq;

-- FINAL SOLUTION
select * from blockdataproject.blockdata.transaction_aggre
gate;
select * from blockdataproject.blockdata.transaction_aggre
gate_cleaned;
```

References & Helps

1. Pub/Sub to Big Query: https://cloud.google.com/dataflow/docs/tutorials/dataflow-stream-to-bigquery
2. Pub/Sub API : https://pkg.go.dev/cloud.google.com/go/pubsub
3. pub/sub to big query video: https://www.youtube.com/watch?v=jXilpXhUXso&ab_channel=SkillCurb
4. CoinGecko Rate Limit problem - https://support.coingecko.com/hc/en-us/articles/4538771776153-What-is-the-rate-limit-for-CoinGecko-API-public-plan