**Real-Time Fraud Detection with GCP (Project 4)**

**Project Goal**

Build a real-time pipeline that:

* Ingests live transaction data using **Pub/Sub**
* Detects **suspicious transactions** in real time using **Dataflow (Apache Beam)**
* Stores flagged results in **BigQuery** for reporting & analytics

**Tech Stack**

| **Tool** | **Purpose** |
| --- | --- |
| **Pub/Sub** | Real-time ingestion |
| **Dataflow** | Stream processing (Apache Beam) |
| **BigQuery** | Flagged fraud transaction storage |
| **Cloud Storage** | Temp storage for staging during pipeline run |
| **Python** | Language used for Beam & publisher scripts |

**Fraud Detection Rules Implemented**

Transactions are flagged as **fraudulent** if:

* amount > 10000 → High amount fraud
* country not in ["Canada", "US"] → Unusual country
* ip starts with "192.168." → Suspicious internal IP

These were chosen as simple fraud heuristics for this prototype.

**Architecture**

User → publisher.py → Pub/Sub → Dataflow (Apache Beam) → BigQuery

**Step-by-Step Implementation**

**1. Enable Required GCP Services**

gcloud services enable pubsub.googleapis.com \

dataflow.googleapis.com \

bigquery.googleapis.com \

storage.googleapis.com \

logging.googleapis.com \

--project=sakina-gcp

**2. Create Pub/Sub Topic**

gcloud pubsub topics create transactions-topic --project=sakina-gcp

**3. Create BigQuery Dataset**

bq mk --dataset sakina-gcp:fraud\_detection

**4. Sample Data (used in the project)**

File: sample\_transactions.json

Example entry:

{

"transaction\_id": "txn004",

"user\_id": "user002",

"amount": 7000,

"country": "USA",

"ip": "192.168.2.2",

"timestamp": "2025-07-18T11:00:00Z"

}

I used this file to simulate real-time transactions published to Pub/Sub.

**5. publisher.py (Used to send transactions to Pub/Sub)**

import json

import time

from google.cloud import pubsub\_v1

project\_id = "sakina-gcp"

topic\_id = "transactions-topic"

publisher = pubsub\_v1.PublisherClient()

topic\_path = publisher.topic\_path(project\_id, topic\_id)

file\_path = "sample\_transactions.json" # Ensure this file is in the same folder

try:

with open(file\_path, "r", encoding="utf-8") as f:

transactions = json.load(f)

except Exception as e:

print(f" Error loading JSON file: {e}")

exit(1)

print(f"🚀 Publishing {len(transactions)} transactions to Pub/Sub topic: {topic\_id}")

for txn in transactions:

try:

message\_json = json.dumps(txn)

message\_bytes = message\_json.encode("utf-8")

future = publisher.publish(topic\_path, data=message\_bytes)

print(f" Published: {message\_json}")

time.sleep(1) # 1 msg/sec to stay within free-tier limits

except Exception as e:

print(f" Failed to publish message: {e}")

print(" Done publishing all transactions.")

**6. pipeline.py (Beam code used for fraud detection)**

import apache\_beam as beam

from apache\_beam.options.pipeline\_options import PipelineOptions, StandardOptions

import json

import logging

class FraudDetectionDoFn(beam.DoFn):

def process(self, element):

try:

record = json.loads(element)

fraud\_reason = None

if record.get("amount", 0) > 10000:

fraud\_reason = "High amount"

elif record.get("country") not in ["Canada", "US"]:

fraud\_reason = "Unusual country"

elif record.get("ip", "").startswith("192.168."):

fraud\_reason = "Suspicious IP"

if fraud\_reason:

record["fraud\_reason"] = fraud\_reason

yield record

except Exception as e:

logging.error(f"Error processing record: {e}")

def run():

import argparse

parser = argparse.ArgumentParser()

parser.add\_argument("--input\_topic", required=True)

parser.add\_argument("--output\_table", required=True)

parser.add\_argument("--project", required=True)

parser.add\_argument("--region", required=True)

parser.add\_argument("--temp\_location", required=True)

args, beam\_args = parser.parse\_known\_args()

options = PipelineOptions(

beam\_args,

project=args.project,

region=args.region,

temp\_location=args.temp\_location,

runner="DataflowRunner", # DataflowRunner for Cloud execution

streaming=True

)

options.view\_as(StandardOptions).streaming = True

with beam.Pipeline(options=options) as p:

(

p

| "ReadFromPubSub" >> beam.io.ReadFromPubSub(topic=args.input\_topic).with\_output\_types(bytes)

| "Decode" >> beam.Map(lambda x: x.decode("utf-8"))

| "DetectFraud" >> beam.ParDo(FraudDetectionDoFn())

| "ToBigQuery" >> beam.io.WriteToBigQuery(

args.output\_table,

schema="transaction\_id:STRING,user\_id:STRING,amount:FLOAT,country:STRING,ip:STRING,timestamp:TIMESTAMP,fraud\_reason:STRING",

write\_disposition=beam.io.BigQueryDisposition.WRITE\_APPEND,

create\_disposition=beam.io.BigQueryDisposition.CREATE\_IF\_NEEDED

)

)

if \_\_name\_\_ == "\_\_main\_\_":

logging.getLogger().setLevel(logging.INFO)

run()

**7. Run the Beam Pipeline to Dataflow**

python pipeline.py \

--input\_topic=projects/sakina-gcp/topics/transactions-topic \

--output\_table=sakina-gcp:fraud\_detection.flagged\_transactions \

--project=sakina-gcp \

--region=us-central1 \

--temp\_location=gs://raw\_bronze1/temp \

--runner=DataflowRunner

This started a streaming job in Dataflow and began writing fraud matches to BigQuery.

**8. Verify in BigQuery**

Check row count:

bq query --use\_legacy\_sql=false '

SELECT COUNT(\*) FROM `sakina-gcp.fraud\_detection.flagged\_transactions`'

Preview flagged data:

bq query --use\_legacy\_sql=false '

SELECT \* FROM `sakina-gcp.fraud\_detection.flagged\_transactions`

ORDER BY timestamp DESC

LIMIT 10'

**Outcome**

* Streamed 10+ transactions to Pub/Sub
* Flagged 5+ as fraud via Beam logic
* Verified results in BigQuery
* Completed full streaming fraud detection prototype

**Optional Cleanup (to avoid costs)**

# Cancel Dataflow

gcloud dataflow jobs list --region=us-central1 --project=sakina-gcp

gcloud dataflow jobs cancel <JOB\_ID> --region=us-central1 --project=sakina-gcp

# Delete topic

gcloud pubsub topics delete transactions-topic --project=sakina-gcp

# Delete BigQuery dataset

bq rm -r -d sakina-gcp:fraud\_detection

# Delete GCS bucket

gsutil rm -r gs://raw\_bronze1