# Customer Account Analysis Data Pipeline

**Bootcamp Project – End-to-End GCP Implementation**  
**Platform:** Google Cloud Platform (GCP)  
**Tools Used:** Draw.io, Cloud Storage, Dataproc (PySpark), BigQuery, Cloud Shell

A diagram of a cloud storage system

AI-generated content may be incorrect.

**Role:** I was responsible for the entire data cleaning and transformation stage in our pipeline. Using PySpark on Google Dataproc, I wrote scripts to ingest 5 raw CSV datasets from Cloud Storage, explicitly defined schemas, handled nulls, and filtered invalid records like negative balances.

After cleaning, I wrote the results in Parquet format into the silver layer of Cloud Storage. I also made sure to avoid schema inference to improve stability and performance. Once the job completed successfully, I shut down the Dataproc cluster to optimize costs. This step was crucial in standardizing the data for the rest of the pipeline.

**Step 1: Data Ingestion – Backend to Raw (Bronze) Bucket**

**Tool:** gsutil  
**Source Bucket:** gs://backend-team-bucket/  
**Target Bucket:** gs://bronze-data-pipeline-455700/

**Files Ingested:**

* accounts.csv
* customers.csv
* loan\_payments.csv
* loans.csv
* transactions.csv

**Command Used:**

gsutil cp gs://backend-team-bucket/\*.csv gs://bronze-data-pipeline-455700/

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2: Data Cleaning – Bronze to Silver Using Dataproc (PySpark)**

**Tool:** Google Colab (initial development) migrated to Dataproc for production processing.

**Script Used:** clean\_all\_files\_dataproc.py  
**Cluster:** my-cluster (deleted after use to save cost)  
**Target Bucket:** gs://silver-data-pipeline-455700/

**Key Steps in the Script:**

* Explicitly define schema (no inferSchema)
* Drop nulls in critical fields
* Filter invalid values (like negative amounts)
* Write to **Parquet format** in silver bucket

**Script Snippet:**

# Accounts cleaning

schema = StructType([

StructField("account\_id", StringType(), False),

StructField("customer\_id", StringType(), False),

StructField("account\_type", StringType(), True),

StructField("balance", DoubleType(), True)

])

df = spark.read.schema(schema).option("header", "true").csv("gs://bronze-data-pipeline-455700/accounts.csv")

df\_cleaned = df.dropna(subset=["account\_id", "customer\_id", "balance"])

df\_cleaned = df\_cleaned.filter(df\_cleaned["balance"] >= 0)

df\_cleaned.write.mode("overwrite").parquet("gs://silver-data-pipeline-455700/accounts")

*Similar logic was applied to: customers, loans, loan\_payments, transactions.*

*A screenshot of a computer

AI-generated content may be incorrect.*

*A screenshot of a computer

AI-generated content may be incorrect.*

*A screenshot of a computer

AI-generated content may be incorrect.*

**Step 3: Join and Aggregate – Silver to Gold**

**Script Used:** join\_and\_aggregate.py  
**Target Bucket:** gs://gold-data-pipeline-455700/customer\_account\_balance/

**Key Logic:**

* Join accounts and customers on customer\_id
* Group by customer\_id to calculate total balance
* Save result as customer\_account\_balance in Gold

**Script Snippet:**

df\_accounts = spark.read.parquet("gs://silver-data-pipeline-455700/accounts")

df\_customers = spark.read.parquet("gs://silver-data-pipeline-455700/customers")

df\_joined = df\_accounts.join(df\_customers, on="customer\_id", how="inner")

df\_agg = df\_joined.groupBy("customer\_id") \

.agg(\_sum("balance").alias("total\_balance"))

df\_agg.write.mode("overwrite").parquet("gs://gold-data-pipeline-455700/customer\_account\_balance/")

A screenshot of a computer

AI-generated content may be incorrect.

**Step 4: BigQuery Load and Daily Upsert**

**Step 4a: Create Final Table in BigQuery**

**Script Used:** create\_customer\_balance\_table.sh

bq query --use\_legacy\_sql=false --project\_id="calcium-field-455700-p2" <<EOF

CREATE TABLE IF NOT EXISTS `calcium-field-455700-p2.customer\_data.customer\_account\_balance` (

customer\_id STRING,

account\_id STRING,

balance NUMERIC,

last\_updated TIMESTAMP

)

OPTIONS(description="Customer Account Balance Table");

EOF

**Step 4b: Load Staging Table (Temporary)**

**Command Used:**

bq load --source\_format=PARQUET --autodetect \

customer\_data.refined\_gold\_customer\_balance\_temp \

gs://gold-data-pipeline-455700/customer\_account\_balance/\*.parquet

**Step 4c: Merge Upsert Script**

**Script Used:** daily\_customer\_balance\_upsert.sh

bq query --use\_legacy\_sql=false --project\_id="calcium-field-455700-p2" <<EOF

MERGE `calcium-field-455700-p2.customer\_data.customer\_account\_balance` T

USING `calcium-field-455700-p2.customer\_data.refined\_gold\_customer\_balance\_temp` S

ON T.customer\_id = S.customer\_id

WHEN MATCHED THEN

UPDATE SET

balance = CAST(S.total\_balance AS NUMERIC),

last\_updated = CURRENT\_TIMESTAMP()

WHEN NOT MATCHED THEN

INSERT (customer\_id, balance, last\_updated)

VALUES (S.customer\_id, CAST(S.total\_balance AS NUMERIC), CURRENT\_TIMESTAMP());

EOF

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 5 (Optional): Simulate Update + Re-run Pipeline**

**Script Used:** modify\_gold\_data.py  
**Purpose:** Modify one row and add one new row in the Gold data to test the upsert

This script was documented but **not executed** (Dataproc cluster was deleted for cost saving).

**Conclusion**

This project demonstrates the successful implementation of an **end-to-end data pipeline on Google Cloud Platform (GCP)** to process and analyze customer account data. The pipeline was designed to move data from raw ingestion to refined insights in BigQuery using a structured, multi-layered architecture.

My key contribution centered on **Data Cleaning**, where I developed a comprehensive PySpark script to clean and validate five raw datasets. I explicitly defined schemas, handled missing and invalid records, and saved the cleaned data in Parquet format to the curated (silver) storage layer.

Throughout the pipeline, I:

* Managed data ingestion from a backend bucket to a raw Cloud Storage layer
* Applied transformation and aggregation logic using **PySpark on Dataproc**
* Wrote refined outputs to Cloud Storage and then loaded them into **BigQuery**
* Designed and executed **MERGE-based upsert logic** for daily updates to the warehouse

This project deepened my practical knowledge of **PySpark, GCP services (Dataproc, Cloud Storage, BigQuery), and shell scripting**, while reinforcing best practices in modular data processing, cloud cost control, and incremental data handling.

With each component functioning as intended, the pipeline is now optimized for daily processing and can be scaled or scheduled using orchestration tools like **Cloud Scheduler** or **Cloud Composer**. This project reflects both technical proficiency and practical design thinking in modern cloud data engineering.