

Customer Data Platform (CDP) ON GCP



Name	Employee ID	Cohort Code
P Suhas Rao	2392975	INTAIA25GCP002

Table of Contents

1. Introduction	3
1.1 About this Document	3
1.1.1 Purpose & Scope of the Document	3
1.2 About the Software System	3
1.2.1 Scope of the System	3
1.2.2 Exclusions	3
1.2.3 System Perspective	3
1.2.4 System Environment	3
1.2.5 Architecture Diagram	4
1.2.6 User Characteristics	4
1.2.7 Impact of the System	4
1.2.8 Assumptions, Risks / Constraints	5
1.2.9 Design Constraints	5
2 System Requirements	5
2.1 Functional Requirements	5
3 Implementation	6
4 Dashboards	26
5 Monitoring and Logging	30
6 Changed Log	33

1. Introduction

1.1 About this Document

1.1.1 Purpose & Scope of the Document

This document presents the implementation details of a Customer Data Platform (CDP) built on Google Cloud Platform (GCP). It outlines the system architecture, design components, data security mechanisms, KPIs, dashboards, and monitoring setup.

1.2 About the Software System

1.2.1 Scope of the System

The system ingests real-time customer and web analytics data, processes and stores it in BigQuery, applies encryption to sensitive fields using KMS, and visualizes business KPIs through interactive dashboards.

1.2.2 Exclusions

This system does not include advanced machine learning modeling, third-party marketing integrations, or external data source connectors beyond CRM and web analytics.

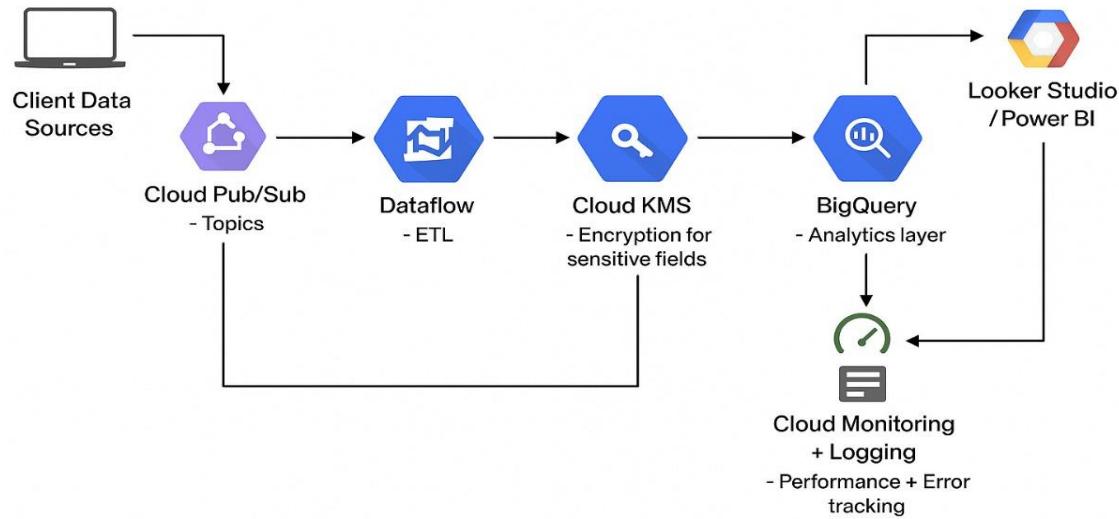
1.2.3 System Perspective

The CDP system is a cloud-native solution integrated into the GCP ecosystem, interacting with services like Pub/Sub, Dataflow, BigQuery, Cloud KMS, IAM, and Looker Studio.

1.2.4 System Environment

- Cloud Platform: Google Cloud Platform (GCP)
- Storage: BigQuery
- Data Ingestion: Cloud Pub/Sub
- Processing: Dataflow
- Security: Cloud KMS, IAM
- Visualization: Looker Studio
- Monitoring: Cloud Monitoring & Logging

1.2.5 Architecture Diagram



(1) High Level Architecture

1.2.6 User Characteristics

- Data Analysts: Use dashboards for business insights
- Engineers: Maintain pipelines and security
- Managers: Track KPIs via dashboards
- Admins: Configure access and monitor performance

1.2.7 Impact of the System

- Real-time insight into customer behavior
- Improved data governance and security
- Centralized data source for marketing and business intelligence

1.2.8 Assumptions, Risks / Constraints

- Assumes stable cloud resource availability
- Risk of pipeline failure if Pub/Sub/Dataflow is misconfigured
- Constraints include cost limitations and dashboard tool scope

1.2.9 Design Constraints

- Must use only GCP-native services
- Sensitive fields must be encrypted before storage
- Dashboards must be built on Looker Studio

2 System Requirements

2.1 Functional Requirements

- Real-time ingestion of CRM and Web Analytics data via Pub/Sub
- Transformation of data using Dataflow
- Secure storage of data in BigQuery
- Encryption of sensitive transaction fields using Cloud KMS
- Implementation of 8 business KPIs using BigQuery
- Creation of 3 Looker Studio dashboards
- Email alerting on BigQuery job errors using Monitoring
- Access control using IAM and VPC Service Controls

3 Implementation

Data Sources

- **CRM Data** (CSV format) – 9 tables -> customers, addresses, demographics, preferences, loyalty program, engagement, customer feedback, customer segments, data login activity.
- **Web Analytics Data** (JSON format) - 5 tables -> click events, conversion events, device information, page views, session data.
- **Transactional Data** (CSV format) – 15 tables -> orders, payments, invoices, shipments, returns, refunds, shipment tracking, order returns, payment transaction, order items, inventory, product catalog, promotions, supplier data, inventory logs.

Note: All data are generated using faker library of python.

Data Ingestion and Storage

Raw data files were initially stored in Google Drive. These files were programmatically transferred to Google Cloud Storage (GCS) Buckets namely crm_sample, web_analy, suhas_transaction using a Python script executed in Google Colab. This automated approach ensured secure and efficient migration of data from Drive to the cloud.

Once the data was available in GCS, it was imported into BigQuery by creating three separate datasets corresponding to the source systems:

CRM

Web

Transaction

Each dataset was structured with appropriate tables reflecting the schema of the original source data. This served as the raw data layer for the analytics workflow.

Google Cloud 22Apr PSuhasRao 6May CTS Search (/) for resources, docs, products, and more

Cloud Storage Buckets + Create C Refresh Go to path Learn

Overview Filter buckets

Buckets Name ↑ Created Location type Location Default storage class Last modified

- crm_sample Apr 23, 2025, 5:23:49 PM Multi-region us Standard Apr 23, 2025, 5:23:49 PM
- suhas_transaction Apr 25, 2025, 3:54:55 PM Multi-region us Standard Apr 25, 2025, 3:54:55 PM
- web_analy Apr 25, 2025, 1:59:29 PM Multi-region us Standard Apr 25, 2025, 1:59:29 PM

Monitoring Settings Storage Intelligence Insights datasets

(2) Buckets

22Apr PSuhasRao 6May CTS Search (/) for resources, docs, products, and more

Bucket details Bucket details Go to path Refresh Learn

Bucket details Bucket details Go to path Refresh Learn

Bucket details Bucket details Go to path Refresh Learn

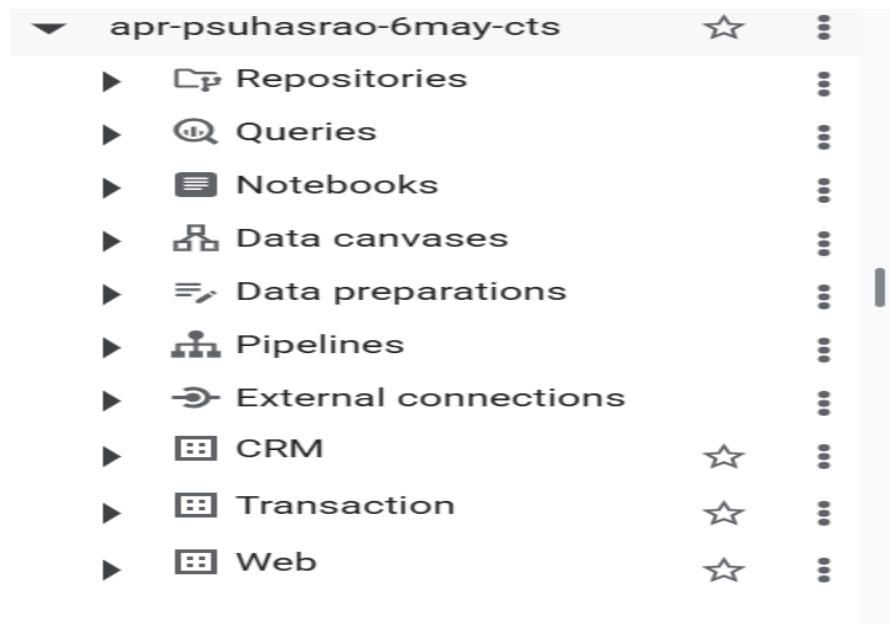
22Apr PSuhasRao 6May CTS Search (/) for resources, docs, products, and more

Bucket details Go to path Refresh Learn

Bucket details Bucket details Go to path Refresh Learn

Bucket details Bucket details Go to path Refresh Learn

(3) Data in respective buckets



(4) BigQuery Datasets

CRM			Web		
datacustomer...	☆	⋮	click_events	☆	⋮
datacustomers1	☆	⋮	conversion_ev...	☆	⋮
datademograp...	☆	⋮	device_info	☆	⋮
dataengagem...	☆	⋮	page_views	☆	⋮
datafeedback1	☆	⋮	session_data	☆	⋮
datalogin_acti...	☆	⋮			
dataloyalty1	☆	⋮			
datapreferenc...	☆	⋮			
datasegments1	☆	⋮			

(5) Tables in CRM

(6) Tables in Web

inventory	☆	⋮
inventory_logs	☆	⋮
invoices1	☆	⋮
order_items	☆	⋮
order_returns1	☆	⋮
orders1	☆	⋮
payment_transactions1	☆	⋮
payments1	☆	⋮
product_catalog	☆	⋮
promotions	☆	⋮
refunds1	☆	⋮
returns1	☆	⋮
shipment_tracking1	☆	⋮
shipments1	☆	⋮
supplier_data	☆	⋮

(7) Tables in Transaction

Real-time Streaming Simulation Using Pub/Sub and Dataflow

To simulate a real-time data streaming pipeline, I utilized Google Cloud Pub/Sub and Dataflow. However, due to the high cost and time involved in streaming the full dataset of 20 lakh records, a scaled-down version was implemented using a sample CSV file (employee.csv) for demonstration purposes.

Pub/Sub Topic Creation and Data Publishing

- A Pub/Sub topic named emp_ex was created.
- A Python script was developed to read the employee.csv file and publish records one by one to the emp_ex topic.
- A delay of 10 seconds was introduced between each record to simulate real-time ingestion behavior.

Streaming Dataflow Job

- An **empty table** was created in BigQuery with the schema corresponding to the structure of the employee data.
- A **Dataflow job** was launched using the "**Pub/Sub to BigQuery**" template.

Input: Pub/Sub topic emp_ex

Output: BigQuery table

- As the script ran, data published to Pub/Sub **was ingested in near real-time** by Dataflow and populated into the BigQuery table.

```

import csv
import time
from google.cloud import pubsub_v1
import json
from datetime import datetime
from datetime import timedelta

# Replace with your GCP project ID and Pub/Sub topic name
project_id = "apr-psushasrao-gmail-cts"
topic_id = "emp_ex"

# Pub/Sub Publisher Client
publisher = pubsub_v1.PublisherClient()
topic_path = publisher.topic_path(project_id, topic_id)

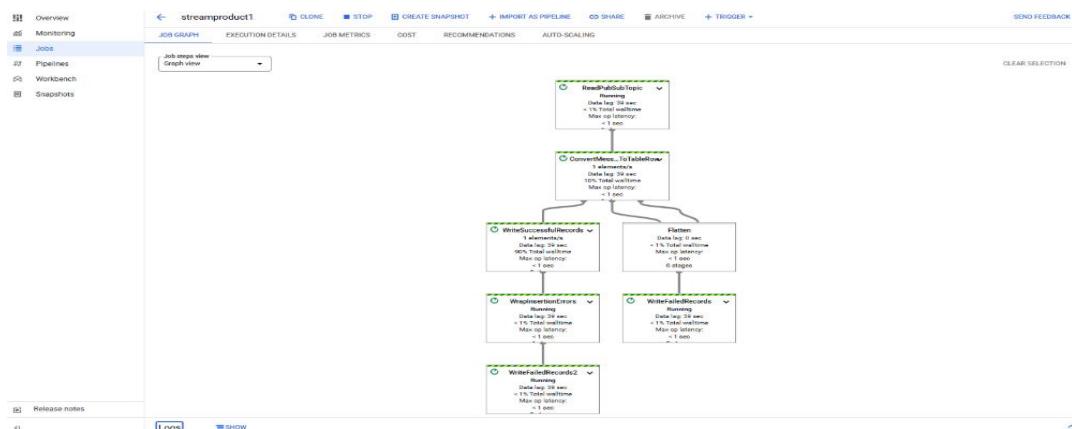
# Function to publish a message to Pub/Sub
def publish_message(empid, firstname, salary):
    timestamp = datetime.utcnow().isoformat() + "Z" # Add current UTC timestamp
    message = {
        "empid": empid,
        "firstname": firstname,
        "salary": salary,
        "timestamp": timestamp,
    }
    # Convert the message to JSON and encode as bytes
    message_json = json.dumps(message).encode("utf-8")
    # Publish the message
    publisher.publish(topic_path, message_json)
    print(f"Published message with ID: {publisher.last_publish_result()}- {message}")

# Path to your CSV file
csv_file_path = "/content/drive/Mydrive/employee.csv"

# Publish records one by one
with open(csv_file_path, mode="r") as file:
    reader = csv.DictReader(file)
    for row in reader:
        empid = row["EmployeeID"]
        firstname = row["FirstName"]
        salary = row["Salary"]
        publish_message(empid, firstname, salary)
        # Wait for 30 seconds before publishing the next record
        time.sleep(30)
Published message with ID: 14671217893092414 - {"empid": "1", "name": "Robert", "salary": "95161", "timestamp": "2025-05-05T06:09:22.128566Z"}
Published message with ID: 1467121590153448 - {"empid": "2", "name": "Linda", "salary": "52274", "timestamp": "2025-05-05T06:09:32.358988Z"}
Published message with ID: 14671217893092414 - {"empid": "3", "name": "John", "salary": "60963", "timestamp": "2025-05-05T06:09:42.392974Z"}

```

(8) Code For Publishing Data to Pub/Sub Topic



(9) Employee Streaming Dataflow Job

The screenshot shows the Google Cloud BigQuery results page for the 'streamproduct1' job. The results table contains the following data:

Row	empid	name	salary	Timestamp
1	1	Robert	95161	2025-04-29 17:07:44.808148 U...
2	2	Linda	52274	2025-04-29 17:09:25.493026 U...
3	3	John	60963	2025-04-29 17:11:05.603072 U...

The page also shows the 'Job history' section at the bottom.

(10) Data Streamed into BigQuery Table

Data Processing with Dataflow (Batch Mode)

Although the intended architecture involved real-time streaming using Pub/Sub and Dataflow, the available data was static. Hence, a batch processing approach was adopted to simulate the ingestion pipeline.

Each source (e.g., Customer, Click Events) included:

- A **JavaScript (.js) transformation file** defining how each record should be parsed and cleaned.
- A **JSON schema file** specifying the structure for the BigQuery table.

The **Cloud Dataflow "Text Files on Cloud Storage to BigQuery" template** was used to launch Dataflow jobs. These jobs:

- Pulled raw data from Cloud Storage.
- Applied the transformation logic using the .js files.
- Loaded the clean and structured data into BigQuery according to the defined .json schema.

This method effectively mimicked a real-time pipeline using batch jobs, ensuring schema enforcement and preprocessing before storage in BigQuery.

The screenshot shows two side-by-side code snippets. On the left is `Customer.js`, a JavaScript file containing a function `transform` that reads a line of CSV data, splits it into values, and creates a JSON object `obj` with various fields like `customer_id`, `first_name`, etc. It also includes helper functions for capitalizing strings, formatting dates/timestamps, and padding numbers. On the right is the generated `Customer.json` BigQuery schema, which is a JSON object mapping field names to their types: `customer_id` (STRING), `first_name` (STRING), `last_name` (STRING), `email` (STRING), `phone_number` (STRING), `gender` (STRING), `date_of_birth` (DATE), `registration_date` (DATE), `status` (STRING), `preferred_language` (STRING), `country` (STRING), `state` (STRING), `city` (STRING), `postal_code` (STRING), and `last_updated` (TIMESTAMP).

```
function transform(line) {
  var values = line.split(',');
  if (values[0] === "customer_id") {
    return null;
  }
  var obj = {};
  obj.customer_id = values[0].trim();
  obj.first_name = capitalize(values[1].trim());
  obj.last_name = capitalize(values[2].trim());
  obj.email = values[3].trim() + "unknown@example.com";
  obj.phone_number = values[4].trim();
  obj.gender = values[5].trim();
  var dob = values[6].trim();
  obj.date_of_birth = formatDate(dob);
  var reg_date = values[7].trim();
  obj.registration_date = formatDate(reg_date);
  obj.status = values[8].trim();
  obj.preferred_language = values[9].trim();
  obj.country = values[10].trim();
  obj.state = values[11].trim();
  obj.city = values[12].trim();
  obj.postal_code = values[13].trim();
  var lastUpdatedRaw = values[14].trim();
  obj.last_updated = formatTimestamp(lastUpdatedRaw);
  return JSON.stringify(obj);
}

function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1).toLowerCase();
}

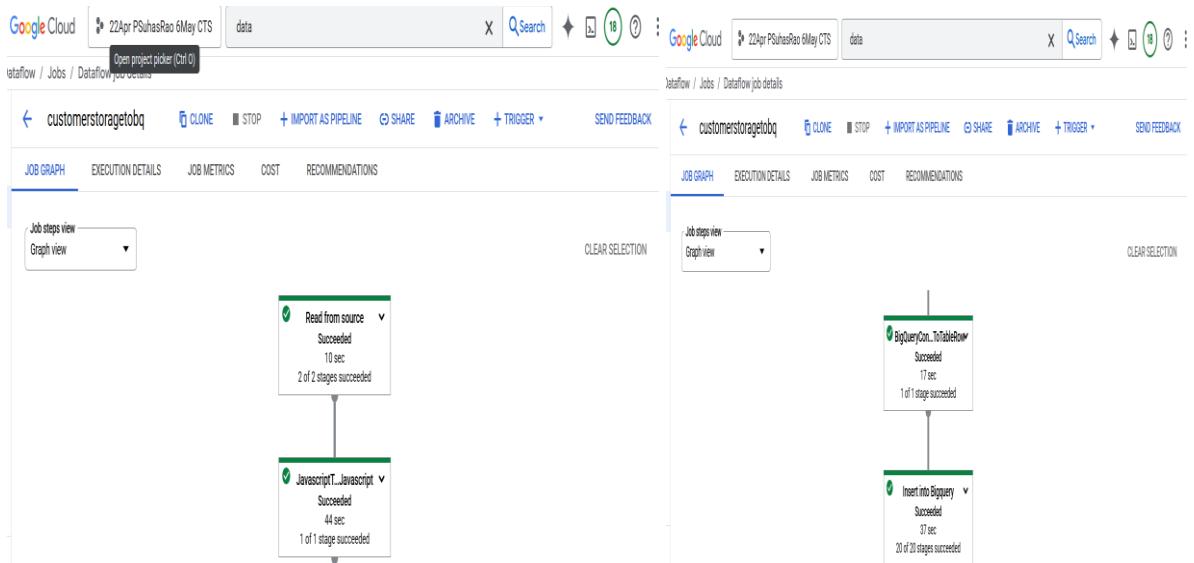
function formatDate(str) {
  var parts = str.split('/');
  if (parts.length === 3) {
    return parts[2] + '-' + pad(parts[1]) + '-' + pad(parts[0]);
  }
  return str;
}

function formatTimestamp(str) {
  if (str.indexOf('T') > -1) {
    var parts = str.split("T");
    var timePart = parts[1].split("Z")[0].split(".").slice(0, 2);
    return parts[0] + " " + timePart[0];
  }
  return str;
}

function pad(n) {
  return n.length === 1 ? '0' + n : n;
}
```

(11) Customer.js

(12) Customer.json



(13) Customer Streaming Dataflow Job

<input type="checkbox"/> Field name	Type	Mode	Key	Collation	Default Value	Policy Tags ?
<input type="checkbox"/> customer_id	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> first_name	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> last_name	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> email	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> phone_number	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> gender	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> date_of_birth	DATE	NULLABLE	-	-	-	-
<input type="checkbox"/> registration_date	DATE	NULLABLE	-	-	-	-
<input type="checkbox"/> status	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> preferred_language	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/> address	STRING	NULLABLE	-	-	-	-

(14) Customer Table Schema

1 SELECT * FROM `apr-psuhasrao-6may-cts.Crm_temp.Customer` LIMIT 100;

✓ Query completed

Query results Save results Open in

Job information	Results	Chart	JSON	Execution details	Execution graph
Row 92	customer_id CUST1977793	first_name Maria	last_name Webb	email janewhite@example.org	
93	CUST0766991	Jessica	Rowe	jessicabeck@example.org	
94	CUST0107578	Michael	White	watsonkristie@example.net	
95	CUST1732257	Rebecca	Pearson	bullocklaura@example.com	
96	CUST1138230	Rebecca	Hill	owilliams@example.org	
97	CUST1346032	John	Lamb	sandra28@example.org	
98	CUST1639055	Cynthia	Marshall	marcus31@example.org	
99	CUST1404915	Jamie	Poole	dharper@example.org	
100	CUST0628466	Tiffany	Dudley	cunninghampaige@example.c	

(15) Transformed Customer Data

```
function transform(jsonMessage) {
  var message = JSON.parse(jsonMessage);
  var obj = {};

  obj.click_id = message.click_id ? message.click_id.trim() : null;
  obj.customer_id = message.customer_id ? message.customer_id.trim() : null;
  obj.element_clicked = message.element_clicked ? message.element_clicked.trim() : null;

  // Convert timestamp if present
  var ts = message.timestamp ? message.timestamp.trim() : null;
  if (ts && ts.indexOf("-") != -1) {
    var parts = ts.split("-");
    var timePart = parts[1].split("T")[0].split(":");
    obj.timestamp = parts[0] + " " + timePart;
  } else {
    obj.timestamp = ts;
  }

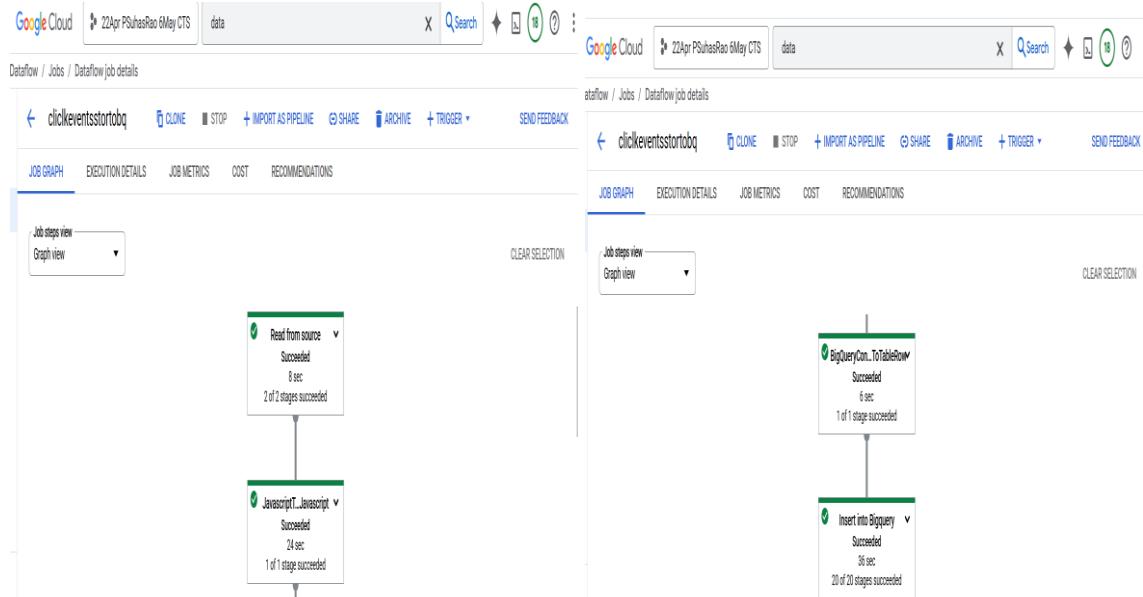
  obj.page_url = message.page_url ? message.page_url.trim() : null;
  obj.click_position_x = message.click_position_x != undefined ? parseInt(message.click_position.x) : null;
  obj.click_position_y = message.click_position_y != undefined ? parseInt(message.click_position.y) : null;
  obj.session_id = message.session_id ? message.session_id.trim() : null;
  obj.device_type = message.device_type ? message.device_type.trim() : null;
  obj.browser = message.browser ? message.browser.trim() : null;
  obj.cta_flag = (typeof message.cta_flag == "boolean") ? message.cta_flag : null;

  return JSON.stringify(obj);
}
```

```
]
} "BigQuery Schema": [
  { "name": "click_id", "type": "STRING" },
  { "name": "customer_id", "type": "STRING" },
  { "name": "element_clicked", "type": "STRING" },
  { "name": "timestamp", "type": "TIMESTAMP" },
  { "name": "page_url", "type": "STRING" },
  { "name": "click_position_x", "type": "INTEGER" },
  { "name": "click_position_y", "type": "INTEGER" },
  { "name": "session_id", "type": "STRING" },
  { "name": "device_type", "type": "STRING" },
  { "name": "browser", "type": "STRING" },
  { "name": "cta_flag", "type": "BOOLEAN" }
]
```

(16) Click_Events.js

(17) Click_Events.json



(18) Click events Streaming Dataflow Job

The image shows the Google Cloud BigQuery schema for the 'clicks' table. The schema is displayed in a table format with columns: Schema, Details, Preview, Table Explorer, Preview (which is highlighted), Insights, Lineage, Data Profile, and Data Quality.

Schema:

Field name	Type	Mode	Key	Collation	Default Value	Policy Tags
click_id	STRING	NULLABLE	-	-	-	-
customer_id	STRING	NULLABLE	-	-	-	-
element_clicked	STRING	NULLABLE	-	-	-	-
timestamp	TIMESTAMP	NULLABLE	-	-	-	-
page_url	STRING	NULLABLE	-	-	-	-
click_position_x	INTEGER	NULLABLE	-	-	-	-
click_position_y	INTEGER	NULLABLE	-	-	-	-
session_id	STRING	NULLABLE	-	-	-	-
device_type	STRING	NULLABLE	-	-	-	-
browser	STRING	NULLABLE	-	-	-	-
cta_flag	BOOLEAN	NULLABLE	-	-	-	-

Actions:

- Buttons: 'Edit schema' and 'View row access policies'.

(19) Click Events Table Schema

22Apr PSuhasRao 6May CTS | big | | Search | 21 | ? | S

Try resources | ?

event_clicks | Query | Open in | Share | Copy | Snapshot | Up | Down

only

Schema Details Preview Table Explorer **Preview** Insights Lineage Data Profile Data Quality

Row	click_id	customer_id	element_clicked	timestamp	pa
1	CLK0979334	CUST0979334	Add to Cart	2025-04-05 21:53:02 UTC	ht
2	CLK0662970	CUST0662970	Add to Cart	2025-03-31 11:00:06 UTC	ht
3	CLK0777786	CUST0777786	Add to Cart	2025-04-10 23:34:46 UTC	ht
4	CLK0887050	CUST0887050	Add to Cart	2025-04-03 22:53:10 UTC	ht
5	CLK0892275	CUST0892275	Add to Cart	2025-03-28 10:27:24 UTC	ht
6	CLK0005911	CUST0005911	Add to Cart	2025-04-20 10:39:08 UTC	ht
7	CLK0156574	CUST0156574	Add to Cart	2025-04-06 03:04:34 UTC	ht
8	CLK0574390	CUST0574390	Add to Cart	2025-04-23 04:56:22 UTC	ht
9	CLK0458672	CUST0458672	Add to Cart	2025-04-19 20:54:42 UTC	ht
10	CLK0216654	CUST0216654	Add to Cart	2025-04-24 16:34:48 UTC	ht
11	CLK0562673	CUST0562673	Add to Cart	2025-03-30 01:32:05 UTC	ht

Results per page: 50 | 1 – 50 of 1000000 | 1 / 20 / Next / Last

(20) Transformed Click Events Data

Encryption of Sensitive fields in Transactional Data

The following sensitive fields were encrypted before storage:

orders1: customer_id, shipping_address, billing_address, payment_method

order_items: supplier_id, supplier_name

payments1: customer_id, payment_method, transaction_id, authorization_code, account_number_masked, payment_ip

invoices1: customer_id, billing_address, payment_method, bank_name, account_number_masked

shipments1: customer_id, tracking_number, origin, destination

returns1: customer_id, return_reason, processed_by

refunds1: customer_id, bank_name, account_number_masked, refund_reason

inventory: supplier_id, batch_number

product_catalog: supplier_id

promotions: promotion_code

payment_transactions1: customer_id, account_number_masked, transaction_reference

shipment_tracking1: customer_id, tracking_number, delivery_address

order_returns1: customer_id, pickup_address

supplier_data: supplier_name, contact_name, contact_email, phone_number, address

inventory_logs: logged_by, approved_by

These fields were identified as containing personally identifiable information (PII) or confidential data and were encrypted to ensure data security during storage and processing.

The screenshot shows the Microsoft Cloud KMS Key Management interface. On the left, there is a sidebar with various navigation options: Cyber insurance hub, Binary Authorisation, Advisory notifications, Access Approval, Managed Microsoft AD, Sensitive data protection, Data Loss Prevention, Certificate Authority Se..., Key Management (which is selected and highlighted in blue), Certificate manager, Secret Manager, Marketplace, and Release notes. The main content area has a header with 'Key ring details', '+CREATE KEY', '+CREATE IMPORT JOB', 'REFRESH', and 'SHOW INFO PANEL'. Below this, there are two tabs: 'KEYS' (selected) and 'IMPORT JOBS'. A sub-section titled 'Keys for transaction-keyring1 key ring' provides a brief description of what a cryptographic key is used for. A table lists the keys in the key ring, with one entry visible: 'transaction-key1' (Status: Available, Protection level: Software, Purpose: Symmetric encrypt/decrypt, Next rotation: 28 Jul 2025). A note at the bottom states 'No keys selected'.

(21) Key Generation in KMS

```

from google.colab import files
!pip install google-cloud-storage google-cloud-kms

uploaded_files.upload()

[Choose Files] No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
having apr-psuhasrao-6may-cts-f6a3dec6e0377.json to apr-psuhasrao-6may-cts-f6a3dec6e0377.json

import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "apr-psuhasrao-6may-cts-f6a3dec6e0377.json"

import asyncio
import base64
import csv
import io
import fastapi_asyncio
from google.cloud import storage, kms_v1
from concurrent.futures import ThreadPoolExecutor

# CONFIGURATION
BUCKET_NAME = "suhas_transaction"
DESTINATION_FOLDER = "Encrypted_Data/"
destination_bucket_name = "encrypted_transaction"
KEY_NAME = "projects/apr-psuhasrao-6may-cts/locations/global/keyRings/transaction-keyring1/cryptoKeys/transaction-key1"

file_list = [
    "orders1", "order_items", "payments1", "invoices1", "shipments1",
    "returns1", "refunds1", "inventory", "product_catalog", "promotions",
    "payment_transactions1", "shipment_tracking1", "order_returns1",
    "supplier_data", "inventory_logs"
]

sensitive_fields = {
    "orders1": ["customer_id", "shipping_address", "billing_address", "payment_method"],
    "order_items": ["supplier_id", "supplier_name"],
    "payments1": ["customer_id", "payment_method", "transaction_id", "authorization_code", "account_number_masked", "payment_ip"],
    "invoices1": ["customer_id", "billing_address", "payment_method", "bank_name", "account_number_masked"],
    "shipments1": ["customer_id", "tracking_number", "origin", "destination"],
    "returns1": ["customer_id", "return_reason", "processed_by"],
    "refunds1": ["customer_id", "batch_number", "account_number_masked", "refund_reason"],
    "inventory": ["supplier_id", "batch_number"],
    "product_catalog": ["supplier_id"],
    "payment_transactions1": ["customer_id", "account_number_masked", "transaction_reference"],
    "shipment_tracking1": ["customer_id", "tracking_number", "delivery_address"],
    "order_returns1": ["customer_id", "pickup_address"],
    "supplier_data": ["supplier_name", "contact_name", "contact_email", "phone_number", "address"],
    "inventory_logs": ["logged_by", "approved_by"]
}

```

```

# Clients
storage_client = storage.Client()
kms_client = kms_v1.KeyManagementServiceClient()

# ThreadPoolExecutor for async operations
executor = ThreadPoolExecutor(max_workers=10)

def encrypt_field(value: str) -> str:
    """Encrypt the field value using KMS"""
    if not value:
        return value
    response = kms_client.encrypt(
        request={"name": KEY_NAME, "plaintext": value.encode("utf-8")}
    )
    return base64.b64encode(response.ciphertext).decode("utf-8")

def process_csv(blob, fields_to_encrypt):
    """Process CSV and encrypt sensitive fields"""
    data = blob.download_as_text()
    input_io = io.StringIO(data)
    output_io = io.StringIO()

    reader = csv.DictReader(input_io)
    writer = csv.DictWriter(output_io, fieldnames=reader.fieldnames)
    writer.writeheader()

    for row in reader:
        for field in fields_to_encrypt:
            if field in row and row[field]:
                row[field] = encrypt_field(row[field])
        writer.writerow(row)

    # Destination path
    file_name = blob.name.split("/")[-1]
    destination_blob_path = f"{DESTINATION_FOLDER}{file_name}"

    # Upload encrypted file to the destination folder
    encrypted_blob = storage_client.bucket(BUCKET_NAME).blob(destination_blob_path)
    encrypted_blob.upload_from_string(output_io.getvalue(), content_type="text/csv")
    print(f"Uploaded {filename} successfully.")

async def process_file_async(blob, fields_to_encrypt):
    """Wrapper for processing CSV asynchronously with a thread pool executor"""
    await asyncio.get_event_loop().run_in_executor(executor, process_csv, blob, fields_to_encrypt)

async def main():
    """Main function to process all files in parallel"""
    bucket = storage_client.bucket(BUCKET_NAME)
    blobs = bucket.list_blobs(prefix=SOURCE_FOLDER)

    tasks = []
    for blob in blobs:
        file_name = blob.name.split("/")[-1]
        if file_name in sensitive_fields:
            print(f"Encrypting: {file_name}")
            tasks.append(process_file_async(blob, sensitive_fields[file_name]))
        else:
            print(f"Skipping: {file_name} (no sensitive fields defined)")

    # Wait for all tasks to finish
    await asyncio.gather(*tasks)

# Run the main async function
if __name__ == "__main__":
    asyncio.run(main())

```

(22) Python Code for Encryption

The screenshot shows a cloud storage interface with a top navigation bar and a main content area. The top bar includes a timestamp '22Apr PSuhasRao 6May CTS', a search bar with placeholder 'big', and various icons for refresh, help, and notifications (18). The main content area is titled 'Bucket details' and shows a list of objects under the folder 'Encrypted_Data/'. The table has columns for Name, Size, Type, and Created. Each row contains a small icon, the object name, its size, type ('text/csv'), and creation date ('Apr 29, 2020'). There are download and more options icons for each row.

	Name	Size	Type	Created		
	inventory	164.7 MB	text/csv	Apr 29, 2020		
	inventory_logs	206.4 MB	text/csv	Apr 29, 2020		
	invoices1	259.7 MB	text/csv	Apr 29, 2020		
	order_items	151.6 MB	text/csv	Apr 29, 2020		
	order_returns1	213.8 MB	text/csv	Apr 29, 2020		
	orders1	310.6 MB	text/csv	Apr 29, 2020		
	payment_transactions1	243.5 MB	text/csv	Apr 29, 2020		
	payments1	367.5 MB	text/csv	Apr 29, 2020		
	product_catalog	146.1 MB	text/csv	Apr 29, 2020		
	promotions	160.6 MB	text/csv	Apr 29, 2020		
	refunds1	309.7 MB	text/csv	Apr 29, 2020		
	returns1	219.2 MB	text/csv	Apr 29, 2020		

(23) Bucket in which Encrypted Data is stored

The screenshot shows the Google Cloud BigQuery interface. The left sidebar lists datasets under 'transaction_encrypt'. The main panel displays the dataset 'transaction_encrypt' with its properties: Created (Apr 29, 2025, 3:22:45 PM UTC+5:30), Default table expiration (Never), Last modified (Apr 29, 2025, 3:22:45 PM UTC+5:30), Data location (US), Description (empty), Default collation (empty), Default rounding mode (ROUNDING_MODE_UNSPECIFIED), Time travel window (7 days), Case insensitive (false), Labels (empty), and Tags (empty). Below this is the 'Dataset replica info' section with Primary location set to US. A 'View replicas' link is also present.

(24) Encrypted Transactional Data in BigQuery

The screenshot shows the Google Cloud BigQuery interface with a query titled 'Untitled query'. The query is: 'SELECT * FROM `apr-psuhasrao-6mav-cts.transaction_encrypt.orders1` LIMIT 10;'. The status is 'Query completed'. The results are displayed in a table with columns: Row, order_id, customer_id, order_date, and order_status. Two rows of data are shown:

Row	order_id	customer_id	order_date	order_status
1	ORD0420621	gAAAAABoEJvxqIX5S6YmiZrn NNf_yj0lfuwaQvKwJrTWhbV V4lltkzmcyl0fa6eUEue_Hcaal 4RVd- ocS1RZEM_FbBCP4cv3yQ==	2025-02-06 11:46:14.561992 U...	Cancelled
2	ORD0173093	gAAAAABoEJvoK8oxXPvJrY KH- z0Fg4VEH4sGRBoUeMluzr zBEo9gvvlmF_SaURcHU4Q 06B9OPXl9qMjhJOPZBdw	2025-04-21 19:03:21.765074 U...	Cancelled

(25) Encrypted Orders Data

22Apr PSuhasRao 6May CTS big

Untitled query

```
1 SELECT * FROM `aor-psuhasrao-6may-cts.transaction_encrypted.orders` LIMIT 10;
```

Query completed

Query results

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	1	shipping_address	billing_address	delivery_date	order_channel
	gAAAAABoEJwD_Ppq2lV_1 gYk8Ga5zBdtWXrUbexqTy MOW1LqFdUSDiQnUjCpk db7K4CSKOcEknb8yKN5v oilodelgFxH8WLHIQ04njT	gAAAAABoEJwXvKv- t2N6Kuns7PjlkP_h1qln6Hi X_Avk9EU2nSuFOHeNshz WUsh6D- 1lKe4z2fGhm0tm5EJbRij5	X_Avk9EU2nSuFOHeNshz WUsh6D- 1lKe4z2fGhm0tm5EJbRij5	2025-02-07 11:46:14.561992 U...	In-Store
	2	gAAAAABoEJv7xxUaDrqj s-aHR8wps2YrzD5Sr1xnDR KJct7CyAm0sXwnd5e4LY nuuMXLv-	gAAAAABoEJwN- waWG4yqv4lXRzOWZwAA 94yF- 3BD4IMXIP3uRJNEqyzuRY aLLHVeA-	2025-04-29 19:03:21.765074 U...	Mobile App

(26) Encrypted Orders Data

22Apr PSuhasRao 6May CTS big

Untitled query

```
1 SELECT * FROM `aor-psuhasrao-6may-cts.transaction_encrypted.order_items` LIMIT 5;
```

Query completed

Query results

Job information	Results	Chart	JSON	Execution details	Execution graph	
Row	1	supplier_id	supplier_name	warranty_months	stock_status	
	gAAAAABoEJxClFqQXLaB7M RENqbTSV7nfQ0_jEqQIBD 0juG7BhGXWlfSbn396LfLlg Z1rcAoqmyjFWdCRTJZmE85A zrmXaa=	gAAAAABoEJxUjZNovskrMnv A7eeggme0cbT- oy9uMpstKKmN7NXQA_DX-N- 2GN8B9Q2TQBu09pj0T8xDq2 O13eHXPrngEcOaQ==		0	Out of Stock	
	2	gAAAAABoEJxCZIFWSbcF DHVMq5UxmRQORUhVmv 130ZKRlo8QPrCryX7amQf 1NpxXC0ZToWV3e- jz4CZRzN8bfJgVeg0ww0h	gAAAAABoEJxU9OlksKNU6 2ghPzQel5- m8_cTcJvxRC0G0pl4W9z 7Yr0VDUJZxGrV90DIaQof xw0wj1bjbHy05Cp-		12	Out of Stock

(27) Encrypted Order Items Data

22Apr PSuhasRao 6May CTS big

Untitled query

```
1 SELECT * FROM `apr-psuhasrao-6may-cts.transaction_encrypted.payments` LIMIT 10;
```

Query completed

Query results

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	1	transaction_id	payment_gateway	authorization_code	payment_date
	gAAAAABoEJybWxT_KnlbSBej AljrsC53Y_eJYqbKo1WCgD_A JOHSITkg2nxv- 3V5o7Gft0eJ7iLo54qYDpCly NtpozD_2QMcw==	PayPal	gAAAAABoEJy9ZOFnhWEBw KNndh_G- nehuNwKhYRmh5Kd4xxOzUtX XOed3oswRx0NtlEx2KWr54jW sHO_z_exJfT57a7-tjR4A==		2025-03-07 09:30:13.7311
	2	gAAAAABoEJybTmtDXwRHUK IB4ghRT6wvP7cXjdc_5zkj35_ 1ChlyOuZ8h68Ew8GKitenJFrF asrMFsw7hiCC12LkZdap_dLI	PayPal	gAAAAABoEJyt5vXElaW72nJ bpKfofu9zMV5- gu9exW8hUb9AMYZ9hWke7D wjqCLhnITFL94r8HjDbzVpMO	2024-12-22 08:33:00.9611

(28) Encrypted Payments Data

IAM and VPC

We were not granted permission to access the VPC at the organizational level.

The screenshot shows the Google Cloud IAM console. The search bar at the top contains 'big'. The main table lists permissions for a Compute Engine default service account and two users ('amit.sharma@iiht.com' and 'suhasingcp@gmail.com'). The 'Allow' tab is selected. The columns are Type, Principal ↑, Name, Role, and Security insights.

Type	Principal ↑	Name	Role	Security insights
930096981212-compute@developer.gserviceaccount.com	Compute Engine default service account	BigQuery Data Editor	Advanced security insight	
		Cloud KMS CryptoKey Decrypter	Advanced security insight	
		Cloud KMS CryptoKey Encrypter	Advanced security insight	
		Dataflow Developer	Advanced security insight	
		Editor		
		Pub/Sub Publisher	Advanced security insight	
		Pub/Sub Subscriber	Advanced security insight	
		Storage Object Viewer	Advanced security insight	
amit.sharma@iiht.com		Owner		
suhasingcp@gmail.com	Suhas Rao	Owner		

(29) Permissions Provided for Service Account

The left panel shows the 'Grant access to "encrypted_transaction"' dialog. It includes sections for Resource ('encrypted_transaction'), Add principals ('priyanshumangal60@gmail.com'), and Assign roles ('Storage Admin'). The right panel shows the 'Bucket details' page for bucket 'buc', listing roles assigned to various principals.

Type	Principal ↑	Name	Role
930096981212-compute@developer.gserviceaccount.com	Compute Engine default service account	Storage Object Viewer	
		Storage Legacy Bucket Owner	
		Storage Legacy Object Owner	
	priyanshumangal60@gmail.com	Storage Admin	

(30) Storage Admin Role

Provided priyanshumangal60@gmail.com a role of storage admin for the bucket 'encrypted_transaction' so that he can access the sensitive data.

(31) Bucket Accessed By priyanshumangal60@gmail.com

Key Performance Indicators (KPI's)

KPI 1: Total Registered Customers

Row	total_customers
1	2000000

(32) KPI 1

KPI 2: Active vs Inactive Customers

Q 2_Active_vs_inactive Run Open in More Save query Download Share Schedule

```
1 SELECT status, COUNT(*) AS count
2 FROM `apr-psuharao-6may-cts.CRM.datacustomers1`
3 GROUP BY status;
```

This query will process 17.17 MB when run.

Query results

Job information Results Chart JSON Execution details Execution graph

Row	status	count
1	Inactive	999406
2	Active	1000594

(33) KPI 2

KPI 3: Total Orders Placed

Q 3_total_orders_placed Run Open in More Save query Download Share Schedule

```
1 SELECT COUNT(*) AS total_orders
2 FROM `apr-psuharao-6may-cts.Transaction.orders1`;
```

This query will process 0 B when run.

Query results

Job information Results Chart JSON Execution details Execution graph

Row	total_orders
1	500000

(34) KPI 3

KPI 4: Total Revenue

Q 4_total_revenue Run Open in ▾ More ▾ Save query ▾ Download Share ▾ Schedule

```
1 SELECT ROUND(SUM(order_amount), 2) AS total_revenue_usd
2 FROM `apr-psuhasrao-6may-cts.Transaction.orders`;
```

This query will process 3.81 MB when run.

Query results Save results ▾ Open in ▾

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	total_revenue_usd				
1	502751119.71				

(35) KPI 4

KPI 5: Average Session Duration

Q 5_Average_session_duration Run Open in ▾ More ▾ Save query ▾ Download Share ▾ Schedule

```
1 SELECT ROUND(AVG(duration_seconds), 2) AS avg_session_duration_secs
2 FROM `apr-psuhasrao-6may-cts.Web.session_data`;
```

This query will process 15.26 MB when run.

Query results Save results ▾ Open in ▾

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	avg_session_duration				
1	1829.73				

(36) KPI 5

KPI 6: Conversion Rate (Last 30 Days)

Q 6_conversion_rate_last_30_days Run Open in More Save query Download Share Schedule

```
1 WITH conversions AS (
2   SELECT COUNT(*) AS conversion_count
3   FROM `apr-psuharsao-6may-cts.Web.conversion_events`
4   WHERE timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
5 ),
6 sessions AS (
7   SELECT COUNT(*) AS session_count
8   FROM `apr-psuharsao-6may-cts.Web.session_data`
9   WHERE session_start >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
10 )
11 SELECT
12   conversions.conversion_count,
13   sessions.session_count,
14   ROUND(SAFE_DIVIDE(conversions.conversion_count, sessions.session_count) * 100, 2) AS conversion_rate_percent
15 FROM conversions, sessions;
```

This query will process 30.52 MB when run.

Query results Save results Open in

Row	conversion_count	session_count	conversion_rate_percent
1	1661012	1660381	100.04

(37) KPI 6

KPI 7: Top Payment Method Used

Q 7_top_payment_method_used Run Open in More Save query Download Share Schedule

```
1 SELECT payment_method, COUNT(*) AS usage_count
2 FROM `apr-psuharsao-6may-cts.Transaction.payments1`
3 GROUP BY payment_method
4 ORDER BY usage_count DESC
5 LIMIT 1;
```

This query will process 4.67 MB when run.

Query results Save results Open in

Row	payment_method	usage_count
1	UPI	100417

(38) KPI 7

KPI 8: Returning Customer Rate (Last 30 Days)

Q 8_returning_customers_lst_30 ...

Run Open in ▾ More ▾ Save query ▾ Download Share ▾ Schedule

```

1 WITH recent_orders AS (
2   SELECT customer_id
3   FROM `apr-psuhasrao-6may-cts.Transaction.orders1`
4   WHERE order_date >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
5 ),
6 repeat_customers AS (
7   SELECT customer_id
8   FROM recent_orders
9   GROUP BY customer_id
10  HAVING COUNT(*) > 1
11 ),
12 all_customers AS (
13   SELECT customer_id
14   FROM recent_orders
15 )
16 
17 SELECT
18   COUNT(DISTINCT repeat_customers.customer_id) AS returning_customers,
19   COUNT(DISTINCT all_customers.customer_id) AS total_customers,
20   ROUND(SAFE_DIVIDE(COUNT(DISTINCT repeat_customers.customer_id), COUNT(DISTINCT all_customers.customer_id)) * 100, 2) AS returning_customer_rate
21 FROM repeat_customers
22 RIGHT JOIN all_customers ON repeat_customers.customer_id = all_customers.customer_id;

```

This query will process 10.01 MB when run.

Query results

Save results Open in ▾

Job information Results Chart JSON Execution details Execution graph

Row	returning_customers	total_customers	returning_customer_rate
1	1125	67388	1.67

(39) KPI 8

4 Dashboards

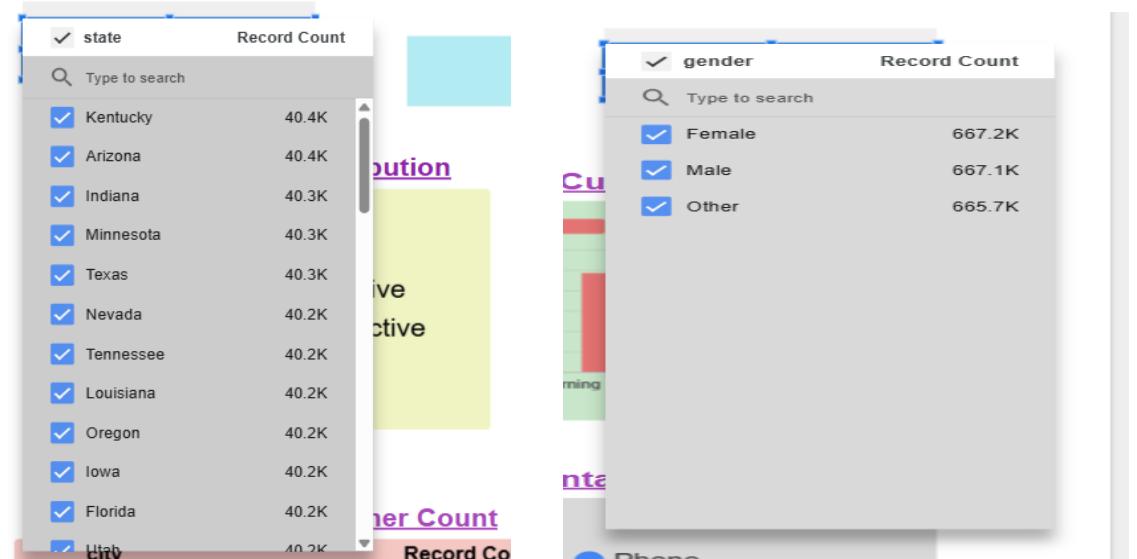
To visualize the processed data effectively, three dashboards were developed:

Customer Overview: Displays key customer metrics including total customers, active users, and customer segmentation.

Web Engagement Analytics: Highlights user interaction patterns such as page views, click events, browsers used etc.

Transaction Insights: Provides visibility into order volumes, payment modes, and revenue trends to support financial analysis.

These dashboards help monitor performance and support data-driven decisions.



(40) Customer Overview Dashboard

Web Engagement Analytics

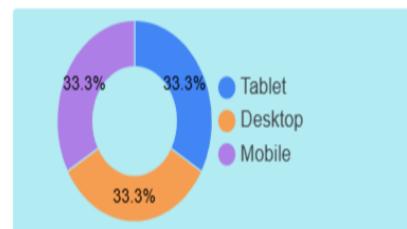
page_url

browser

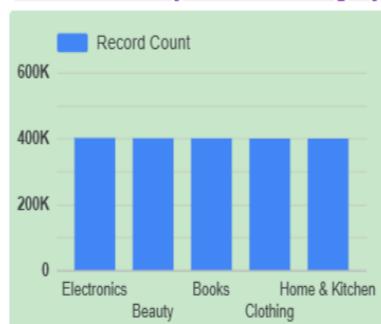
Clicks by Element



User Devices Distribution



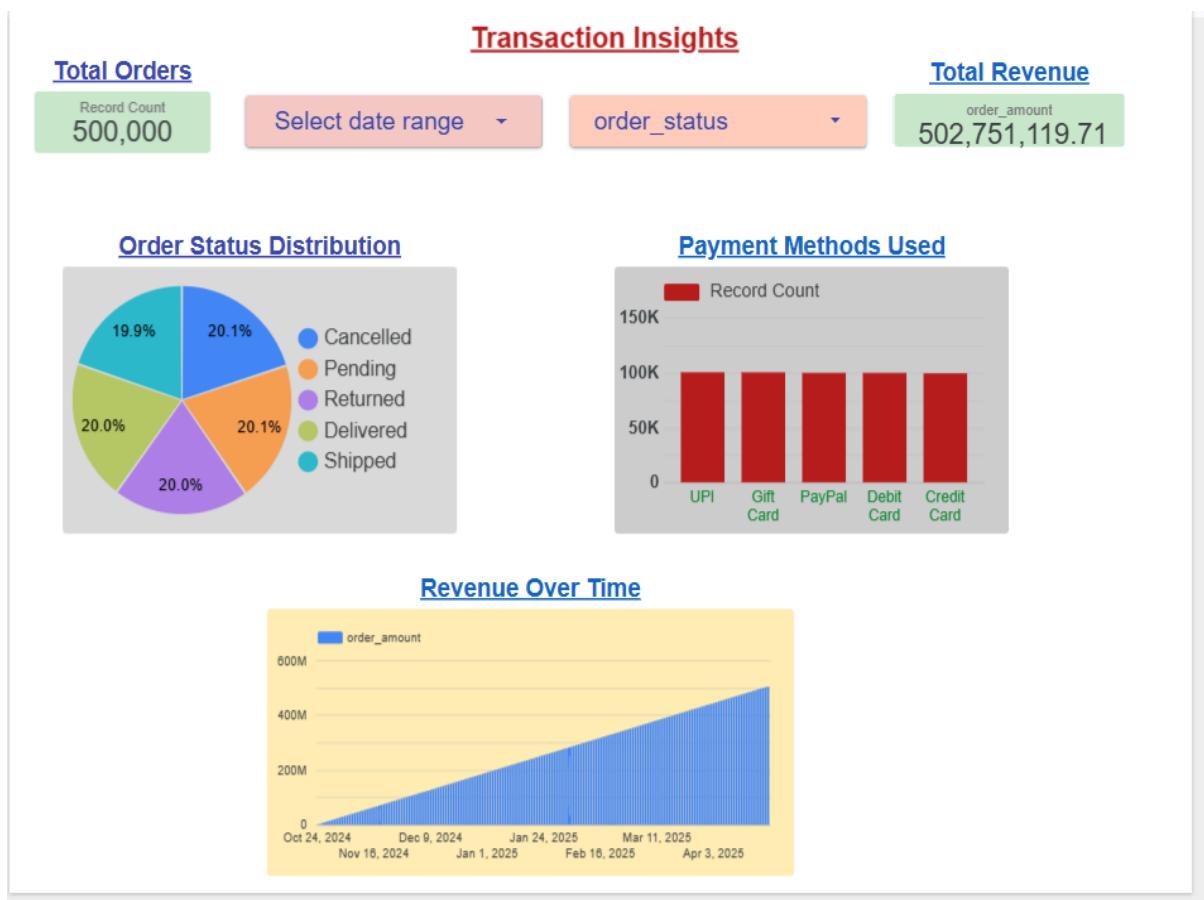
Conversions by Product Category



page_url	Record Count
http://www.smith.co...	7
https://miller.com/w...	6
https://www.smith.c...	6
https://brown.com/...	6
https://www.smith.c...	6
http://www.smith.co...	6
http://johnson.com/...	6
http://johnson.com/...	6
http://www.smith.co...	5
http://www.brown.c...	5
https://www.brown....	5
https://smith.com/w...	5

browser	Record Count
Safari	250.2K
Edge	250.1K
Firefox	250K
Chrome	249.7K

(41) Web Analytics Dashboard



Total Orders

Fixed

Start Date: JAN 2025 ▾ End Date: MAY 2025 ▾

S	M	T	W	T	F	S	S	M	T	W	T	F	S
JAN	1	2	3	4			MAY	1	2	3			
5	6	7	8	9	10	11	4	5	6	7	8	9	10
12	13	14	15	16	17	18	11	12	13	14	15	16	17
19	20	21	22	23	24	25	18	19	20	21	22	23	24
26	27	28	29	30	31		25	26	27	28	29	30	31

Cancel Apply

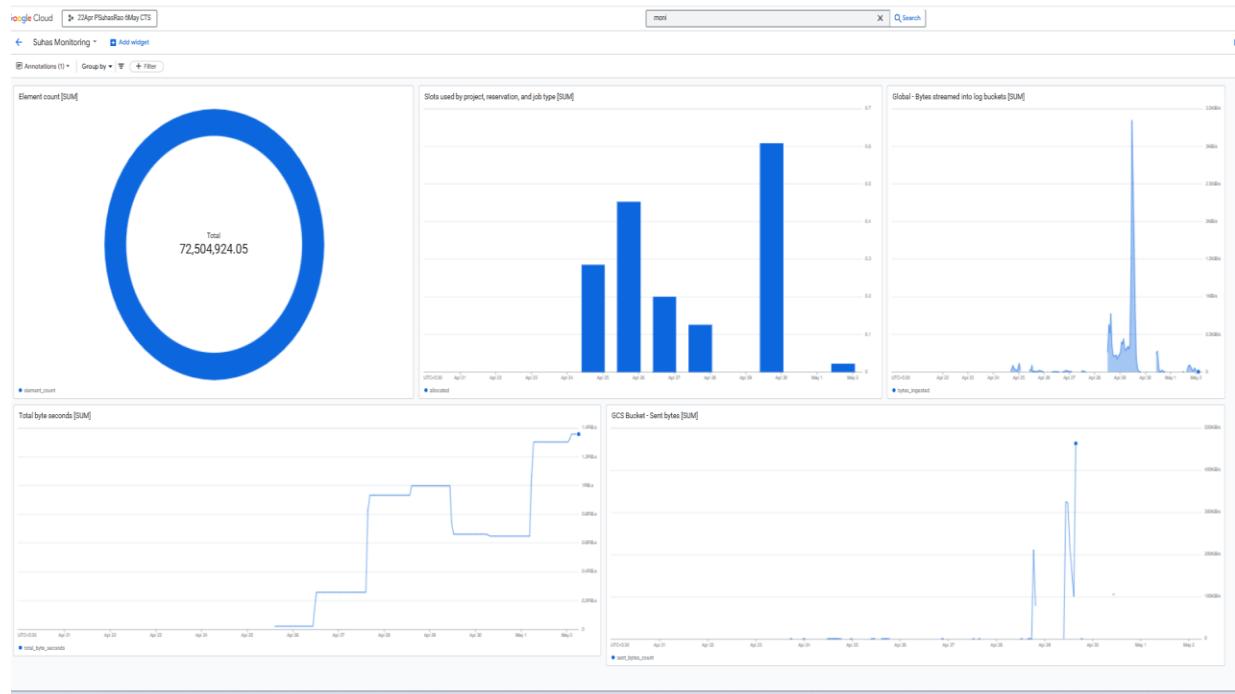
order_status Record Count

Type to search

order_status	Record Count
Cancelled	58.7K
Pending	58.5K
Returned	58.3K
Delivered	58.1K
Shipped	58.1K

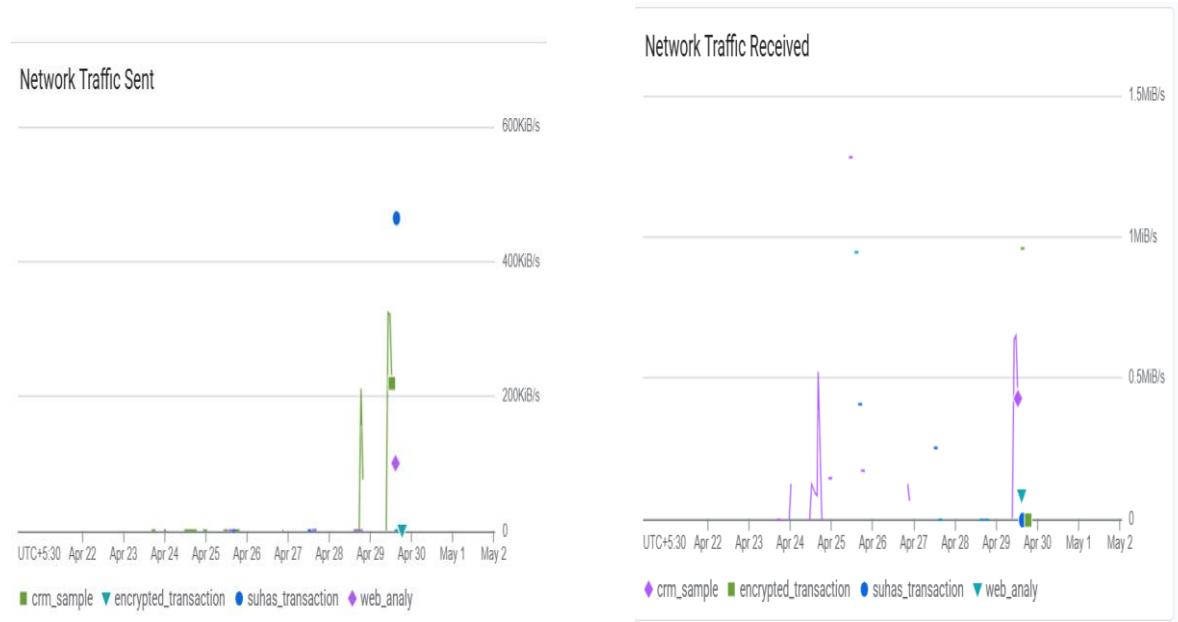
(42) Transaction Insights Dashboard

5 Monitoring and Logging



(43) Monitoring Dashboard

Google Cloud Storage Monitoring





(44) Cloud Storage Monitoring

An alert will be sent to my email address suhasraohub@gmail.com whenever error is triggered in bigquery project.

The screenshot shows the Google Cloud Logs Explorer interface. The top navigation bar includes 'Logs Explorer', 'Query library', 'Share link', 'Preferences', and date/time controls ('Apr 20, 10:18 AM - May 2, 12:42 AM'). A search bar is present at the top right. Below the header, there's a dropdown for 'Project logs' and a search bar for 'Search all fields'. A 'BigQuery Project' dropdown is open, showing 'All log names' and 'Error+' filters. A 'Correlate by' button is also visible. On the left, a 'Fields' sidebar lists 'Severity', 'Time', and 'Summary'. The main area displays a timeline from April 20 to May 2, 2025, with a red bar indicating error events. A specific event on April 20 at 11:43:18 is highlighted with a blue dot. The summary for this event is: SEVERITY: ERROR, TIME: 2025-04-29 11:43:18.839, SUMMARY: bqjob_0d3f2bb_00001_00000-7, from 'bqjob_0d3f2bb_00001_00000-7' at 'bqjob_0d3f2bb_00001_00000-7'. The URL 'io.cloud.google.com/project:apr-psuhasrao-6may-cty.googleapis.com' is shown at the bottom.

Alert details

Provide a name and description for this log alert.

Name BigQuery Error in US

Choose logs to include in the alert

Create an inclusion filter to determine which logs are included in the alert.

Alert query

```
severity>=ERROR  
resource.type="bigquery_project"  
resource.labels.location="US"
```

Set notification frequency and autoclose duration

Configure the minimum amount of time between receiving notifications for logs that match this filter, and the duration to autoclose corresponding incidents.

Time between notifications 1 hr

Incident autoclose duration 7 days

Who should be notified? (optional)

When alerting policy violations occur, you will be notified via these channels.

Notification Channels

- [subhasraobub@googlegroups.com](#)

Notification channels

PagerDuty Services

No PagerDuty services configured

PagerDuty Sync

No PagerDuty Sync channels configured

Slack

No Slack channels configured

Webhooks

No webhook channels configured

Email

No emails configured

SMS

No SMS channels configured

Create Email Channel

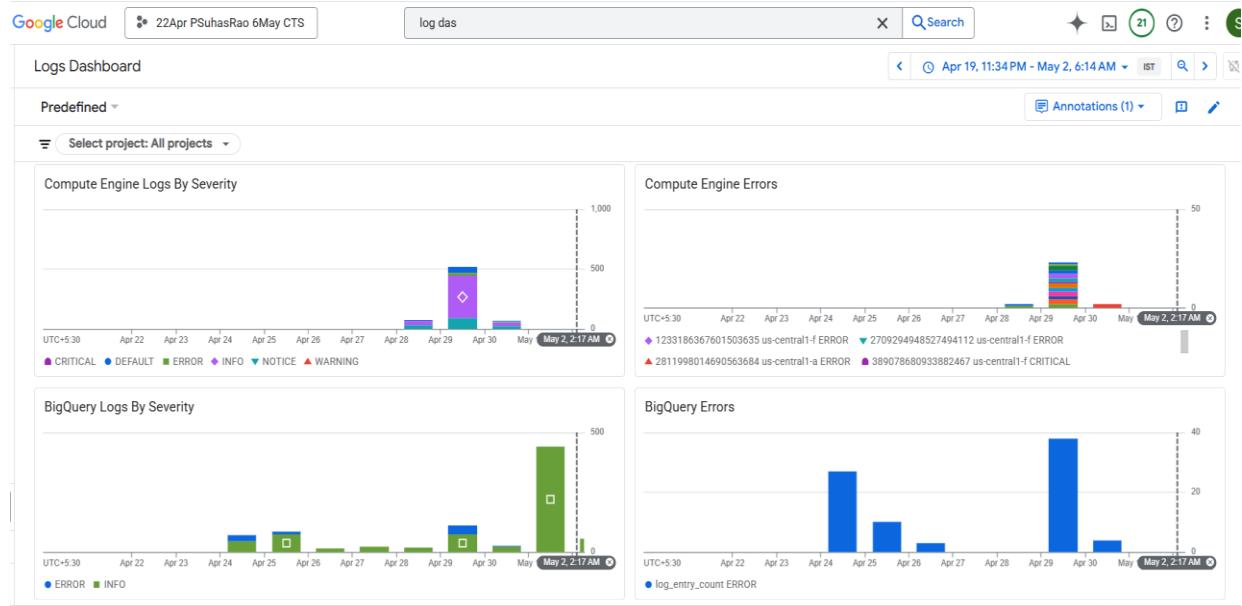
Email addresses can be set to receive notifications from your alerting when a new incident is created.

Email Address *

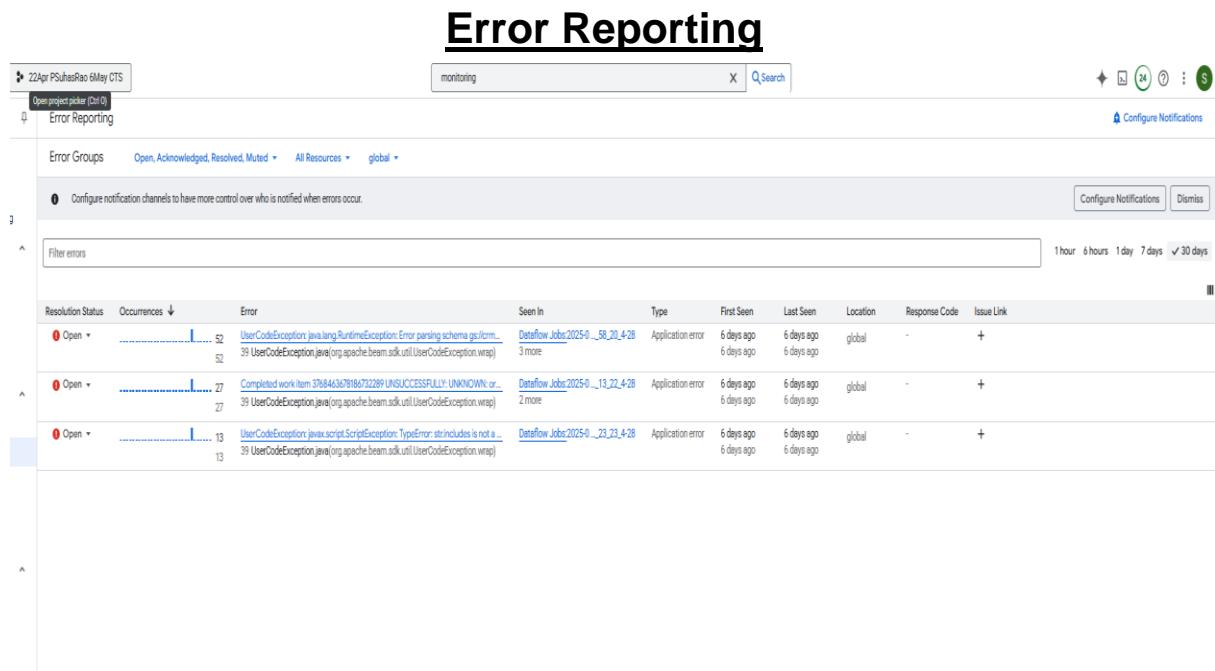
Display Name *

Cancel **Save**

(45) Alerting about BigQuery Project Error



(46) Logging Dashboard



(47) Error Reporting

6 Changed Log

Version	Date	Description
1.0	2025-05-04	Defined the purpose and scope of the document. Detailed system objectives and boundaries. Documented system scope, perspective, and architecture. Outlined functional requirements and covered implementation including monitoring and logging