**ETL Pipeline Project in GCP**

**Create two instances in GCP, one for Cloud Composer and One for Data Fusion**

**Data Fusion:** No Code ETL Solution Provided by Google, do transformation with just drag and drop. In this we have Wrangler, Integrate, Discover and govern, Monitor and Manage. Wrangler is used for data transformation. In this we have multiple connectors, but we will use GCS where we have all the buckets and we will use our bucket and pick the csv file.

Data Fusion is like Power Query Editor, we can mask the sensitive data her too. And then create batch pipeline

In the pipepline we have give the connectors like where we want to sink the data here we are used bigquery. Now at backend it is using dataproc and creating instance there you can check

If you're using **gcloud auth application-default login**, you don't need to specify a service account JSON file explicitly. The Google Cloud SDK will use your authenticated credentials automatically.

Here’s how you can update your script to generate dummy data with passwords and upload it to Google Cloud Storage (GCS) without specifying a service account key file.

**Step 1: Authenticate with GCP**

Run the following command in your terminal to authenticate:

bash

**gcloud auth application-default login**

This will authenticate your local environment to access GCP services.

**Step 2: Install Required Python Libraries**

If you haven't installed them yet, install the required dependencies:

bash

**pip install faker pandas google-cloud-storage**

**Step 3: Generate Dummy Data with Passwords**

Modify your script to generate dummy data, including a password:

python

**import pandas as pd**

**from faker import Faker**

**import random**

**fake = Faker()**

**# Define the number of records**

**num\_records = 1000**

**# Generate dummy data**

**data = []**

**for \_ in range(num\_records):**

**data.append({**

**"id": fake.uuid4(),**

**"name": fake.name(),**

**"email": fake.email(),**

**"password": fake.password(length=12, special\_chars=True, digits=True, upper\_case=True, lower\_case=True),**

**"address": fake.address(),**

**"phone\_number": fake.phone\_number(),**

**"date\_of\_birth": fake.date\_of\_birth(minimum\_age=18, maximum\_age=80).isoformat(),**

**"salary": round(random.uniform(30000, 100000), 2),**

**"company": fake.company(),**

**"job\_title": fake.job(),**

**})**

**# Convert to DataFrame**

**df = pd.DataFrame(data)**

**# Save to CSV**

**csv\_filename = "dummy\_data\_with\_passwords.csv"**

**df.to\_csv(csv\_filename, index=False)**

**print(f"Dummy data saved as {csv\_filename}")**

**Step 4: Upload CSV to Google Cloud Storage**

Since you're using gcloud auth application-default login, you don’t need to specify the service account key. Update your upload function:

python

**from google.cloud import storage**

**# Set your Google Cloud Storage bucket name**

**BUCKET\_NAME = "your-gcs-bucket-name"**

**DESTINATION\_BLOB\_NAME = "dummy\_data\_with\_passwords.csv"**

**# Initialize GCS client (uses default application credentials)**

**storage\_client = storage.Client()**

**# Upload file to GCS**

**def upload\_to\_gcs(bucket\_name, source\_file\_name, destination\_blob\_name):**

**bucket = storage\_client.bucket(bucket\_name)**

**blob = bucket.blob(destination\_blob\_name)**

**blob.upload\_from\_filename(source\_file\_name)**

**print(f"File {source\_file\_name} uploaded to gs://{bucket\_name}/{destination\_blob\_name}.")**

**# Call the function**

**upload\_to\_gcs(BUCKET\_NAME, csv\_filename, DESTINATION\_BLOB\_NAME)**

**Step 5: Verify in Google Cloud Storage**

1. Go to Google Cloud Console.
2. Navigate to **Storage** → **Your Bucket**.
3. Ensure dummy\_data\_with\_passwords.csv is uploaded.

What is cloud composer

**ChatGPT said:**

**Cloud Composer: Managed Apache Airflow on GCP**

**Google Cloud Composer** is a **fully managed workflow orchestration service** built on **Apache Airflow**. It is used for creating, scheduling, and monitoring **ETL pipelines**, **data workflows**, and **automation tasks** in **Google Cloud Platform (GCP)**.

**Key Features of Cloud Composer**

🔹 **Managed Apache Airflow** → Google handles Airflow setup, scaling, and maintenance.  
🔹 **Integration with GCP Services** → Works with **BigQuery, Cloud Storage, Dataflow, Dataproc, Pub/Sub**, etc.  
🔹 **Workflow Automation** → Automates complex workflows for data engineering and ML pipelines.  
🔹 **Scalability** → Dynamically scales Airflow workers based on workload.  
🔹 **Security & Monitoring** → Supports IAM roles, VPC networking, and logs in **Cloud Logging & Monitoring**.

**Use Cases of Cloud Composer**

✅ **ETL & Data Pipelines** → Automate data ingestion, transformation, and loading into **BigQuery**.  
✅ **Machine Learning Pipelines** → Orchestrate ML model training and inference workflows.  
✅ **Data Warehousing** → Automate data movement between **Cloud Storage, BigQuery, and Dataproc**.  
✅ **Cross-Cloud Orchestration** → Manage workflows that run across AWS, Azure, and GCP.  
✅ **CI/CD for Data Workflows** → Automate deployment of data pipelines in **Airflow DAGs**.

**How Cloud Composer Works**

1. **Create a Cloud Composer environment** in GCP.
2. **Write Airflow DAGs (Directed Acyclic Graphs)** → Python scripts defining workflows.
3. **Store DAGs in a Cloud Storage bucket** → Composer automatically picks them up.
4. **Airflow executes tasks based on schedules & dependencies**.
5. **Monitor workflows in Airflow UI** or **Cloud Logging**.

**Example: DAG in Cloud Composer (Airflow)**

python

CopyEdit

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from datetime import datetime

# Define a simple Python function

def print\_hello():

print("Hello from Cloud Composer!")

# Define DAG

default\_args = {"start\_date": datetime(2024, 1, 1)}

dag = DAG("my\_first\_dag", default\_args=default\_args, schedule\_interval="@daily")

# Create a task

task = PythonOperator(

task\_id="hello\_task",

python\_callable=print\_hello,

dag=dag,

)

task # Execute task

**Cloud Composer vs Other Workflow Tools**

| **Feature** | **Cloud Composer (Airflow)** | **Cloud Dataflow** | **Apache NiFi** |
| --- | --- | --- | --- |
| **Use Case** | Orchestration & ETL | Streaming & Batch Processing | Data Ingestion |
| **Technology** | Apache Airflow | Apache Beam | Apache NiFi |
| **Execution Type** | Task-based Workflows | Streaming & Batch | Data Flows |
| **Integrations** | GCP, AWS, On-Prem | BigQuery, Pub/Sub | Cloud & On-Prem |
| **Flexibility** | High | Medium | Low |

**When to Use Cloud Composer?**

✅ You need **complex workflows with dependencies**  
✅ You want **event-driven or scheduled automation**  
✅ You require **GCP-native Airflow orchestration**  
✅ You prefer **Python-based workflow definitions (Airflow DAGs)**

🚀 **Best for Data Engineers & ML Engineers managing multi-step workflows!**

Let me know if you need a hands-on guide for Cloud Composer setup! 🎯

**You said:**

What is data fusion

**ChatGPT said:**

**Cloud Data Fusion: Managed Data Integration on GCP**

**Google Cloud Data Fusion** is a **fully managed, cloud-native ETL (Extract, Transform, Load) and data integration service** built on **CDAP (Cask Data Application Platform)**. It enables **code-free** and **code-based** data pipeline development for moving, transforming, and processing data across different sources.

**Key Features of Cloud Data Fusion**

🔹 **Visual Pipeline Designer** → Drag-and-drop interface for building ETL pipelines.  
🔹 **Pre-built Connectors** → Integrates with **BigQuery, Cloud Storage, Dataproc, Pub/Sub, SQL databases, APIs, and on-prem data sources**.  
🔹 **Hybrid & Multi-Cloud Support** → Connects **on-prem, AWS, Azure, and GCP** data sources.  
🔹 **Batch & Streaming Data Processing** → Uses **Apache Spark** for batch and **Apache Kafka** for streaming.  
🔹 **Scalability & Performance** → Built on **Google Kubernetes Engine (GKE)** for elastic scaling.  
🔹 **Security & Governance** → Supports IAM roles, **VPC networking**, and **Data Catalog integration**.

**Use Cases of Cloud Data Fusion**

✅ **ETL & Data Warehousing** → Ingest and transform raw data for **BigQuery** analytics.  
✅ **Data Lake Ingestion** → Move structured/unstructured data to **Cloud Storage** or **Dataproc**.  
✅ **Streaming & Batch Pipelines** → Real-time event processing with **Pub/Sub & Kafka**.  
✅ **Data Migration** → Move data from on-prem or other clouds to **GCP**.  
✅ **ML Data Preparation** → Clean and transform data before model training in **Vertex AI**.

**How Cloud Data Fusion Works**

1. **Create a Cloud Data Fusion instance** in GCP.
2. **Design a pipeline** using the drag-and-drop UI or write a pipeline in **CDAP API**.
3. **Select data sources & transformations** (e.g., filtering, joins, aggregations).
4. **Execute & Monitor pipelines** → Run pipelines in **Cloud Dataproc (Apache Spark)**.
5. **Store transformed data** in **BigQuery, Cloud Storage, or databases**.

**Example: Data Pipeline in Cloud Data Fusion**

* **Source**: MySQL
* **Transformations**: Data Cleaning, Aggregation
* **Destination**: BigQuery

📌 **Steps**:  
1️⃣ **Add Source** → Drag **MySQL** Connector  
2️⃣ **Apply Transformations** → Use **Data Wrangling**  
3️⃣ **Add Destination** → Select **BigQuery**  
4️⃣ **Run the Pipeline** 🚀

**Cloud Data Fusion vs Other ETL Tools**

| **Feature** | **Cloud Data Fusion** | **Cloud Composer (Airflow)** | **Dataflow (Apache Beam)** |
| --- | --- | --- | --- |
| **Use Case** | ETL & Data Integration | Workflow Orchestration | Real-time & Batch Data Processing |
| **Technology** | CDAP, Spark | Apache Airflow | Apache Beam |
| **Execution Type** | Drag-and-Drop & Code | DAG-based Tasks | Streaming & Batch |
| **Best For** | ETL Pipelines | Task Orchestration | Real-time Data Processing |

**When to Use Cloud Data Fusion?**

✅ **If you want a no-code ETL tool** with a **drag-and-drop UI**  
✅ **If you need pre-built connectors** for easy **data movement & transformation**  
✅ **If you want an alternative to Apache NiFi or Talend**

🚀 **Best for Data Engineers & Analysts who need fast, scalable ETL without coding!**

Let me know if you need help setting up a Data Fusion pipeline! 🎯

**You said:**

What is Data Flow

**ChatGPT said:**

**Cloud Dataflow: Fully Managed Stream & Batch Data Processing on GCP**

**Google Cloud Dataflow** is a **fully managed service** for **streaming and batch data processing** using **Apache Beam**. It enables **real-time and batch data transformation, ETL, and analytics** at scale without managing infrastructure.

**Key Features of Cloud Dataflow**

🔹 **Unified Batch & Streaming Processing** → Processes both real-time and historical data.  
🔹 **Auto-scaling & Serverless** → No need to manage resources; scales automatically.  
🔹 **Powered by Apache Beam** → Write once, run anywhere (GCP, AWS, On-Prem).  
🔹 **Low Latency & High Throughput** → Suitable for big data & real-time applications.  
🔹 **Seamless GCP Integration** → Works with **BigQuery, Cloud Storage, Pub/Sub, AI Platform, Dataproc**, etc.  
🔹 **Fault Tolerance & Monitoring** → Auto-recovery, logging, and Stackdriver monitoring.

**Use Cases of Cloud Dataflow**

✅ **Real-time Stream Processing** → Process live data from **Pub/Sub, Kafka, IoT sensors**.  
✅ **ETL & Data Transformation** → Extract from **Cloud Storage, transform, and load into BigQuery**.  
✅ **Data Lake Pipelines** → Move and clean data from multiple sources into **Cloud Storage**.  
✅ **Fraud Detection** → Analyze transactions in real-time for anomalies.  
✅ **Machine Learning Pipelines** → Preprocess and clean data before ML model training.  
✅ **Log & Event Analytics** → Process server logs, security events, and application logs.

**How Cloud Dataflow Works**

1️⃣ **Write an Apache Beam pipeline** in **Python or Java**.  
2️⃣ **Read data from a source** (e.g., Pub/Sub, Cloud Storage, BigQuery).  
3️⃣ **Apply transformations** (filtering, aggregation, joins).  
4️⃣ **Write data to a destination** (e.g., BigQuery, Cloud Storage).  
5️⃣ **Run the pipeline** in **Cloud Dataflow** → Auto-scales & optimizes execution.

**Example: Streaming Data Processing in Cloud Dataflow**

📌 **Use Case**: Process live messages from **Pub/Sub** and store them in **BigQuery**.

**1️⃣ Install Apache Beam SDK**

bash

CopyEdit

pip install apache-beam[gcp]

**2️⃣ Python Code for Dataflow Pipeline**

python

CopyEdit

import apache\_beam as beam

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

# Define pipeline options

pipeline\_options = PipelineOptions(

runner="DataflowRunner",

project="your-gcp-project-id",

region="us-central1",

temp\_location="gs://your-bucket/temp"

)

# Define the pipeline

with beam.Pipeline(options=pipeline\_options) as p:

(

p

| "Read from Pub/Sub" >> beam.io.ReadFromPubSub(subscription="projects/your-project-id/subscriptions/your-subscription")

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

| "Write to BigQuery" >> beam.io.WriteToBigQuery(

"your-project-id:your\_dataset.your\_table",

schema="message:STRING",

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

)

)

**3️⃣ Run the Pipeline in Dataflow**

bash

CopyEdit

python my\_dataflow\_pipeline.py

**Cloud Dataflow vs Other GCP Services**

| **Feature** | **Cloud Dataflow** | **Cloud Data Fusion** | **Cloud Composer (Airflow)** |
| --- | --- | --- | --- |
| **Use Case** | Real-time & Batch Processing | ETL & Data Integration | Workflow Orchestration |
| **Technology** | Apache Beam | CDAP (Cask Data) | Apache Airflow |
| **Execution Type** | Continuous Streaming & Batch | Drag-and-Drop ETL | DAG-based Task Orchestration |
| **Best For** | Real-time & large-scale processing | No-code ETL pipelines | Scheduling & automation |

**When to Use Cloud Dataflow?**

✅ **If you need real-time data processing (e.g., logs, IoT, event streaming)**  
✅ **If you need large-scale ETL with complex transformations**  
✅ **If you want a fully managed, auto-scaling solution**  
✅ **If you prefer writing Apache Beam pipelines in Python or Java**

🚀 **Best for Data Engineers & Developers handling real-time analytics & big data!**

Let me know if you need help setting up Dataflow pipelines! 🎯

**You said:**

What is Bigquery and Cloud Storage

**ChatGPT said:**

**BigQuery vs Cloud Storage: Key Differences and Use Cases**

**Google BigQuery** and **Google Cloud Storage** are both **data storage solutions** on GCP, but they serve different purposes.

| **Feature** | **BigQuery (Data Warehouse)** | **Cloud Storage (Object Storage)** |
| --- | --- | --- |
| **Purpose** | **Analytics & Querying** | **File/Object Storage** |
| **Data Type** | **Structured (Tables, SQL-based)** | **Unstructured (Images, Videos, CSVs, Backups, etc.)** |
| **Use Case** | Big Data Analysis, BI, ML | Storing and serving large files |
| **Querying** | SQL-based queries | No built-in querying (must move to BigQuery for analysis) |
| **Speed** | Optimized for fast analytics | Optimized for scalable storage |
| **Best For** | **BI, ETL, Machine Learning, Data Science** | **Backup, Archive, Data Lakes, Data Sharing** |

**What is BigQuery?**

🔹 **Fully managed** serverless **data warehouse** for **big data analytics**  
🔹 **SQL-based** queries to process **petabytes of data** in seconds  
🔹 Supports **real-time streaming** & **machine learning (BigQuery ML)**  
🔹 **Optimized for analytics**, not general storage

**BigQuery Use Cases**

✅ Business Intelligence (BI) & Analytics  
✅ Machine Learning (BigQuery ML)  
✅ Real-time Data Processing (via Pub/Sub)  
✅ ETL & Data Warehouse

📌 **Example Query in BigQuery**

sql

CopyEdit

SELECT state, AVG(sales) as avg\_sales

FROM `your\_project.dataset.sales\_data`

GROUP BY state

ORDER BY avg\_sales DESC;

**What is Cloud Storage?**

🔹 **Fully managed** object storage for **files, backups, and large datasets**  
🔹 **Stores any data type** (structured & unstructured)  
🔹 Supports **data lakes** and integrates with **BigQuery**  
🔹 **Cheaper than BigQuery for storage**, but no direct SQL querying

**Cloud Storage Use Cases**

✅ Store **images, videos, backups, logs**  
✅ Create **Data Lakes** for **BigQuery or Dataproc**  
✅ **Serve static websites** and **ML datasets**  
✅ **Long-term archiving** (Coldline & Archive Storage)

📌 **Example: Upload a File to Cloud Storage**

bash

CopyEdit

gsutil cp local\_file.csv gs://your-bucket/

📌 **Example: Move Data from Cloud Storage to BigQuery**

bash

CopyEdit

bq load --source\_format=CSV dataset.sales\_data gs://your-bucket/sales\_data.csv

**When to Use BigQuery vs Cloud Storage?**

✅ **Use BigQuery if you need SQL-based analytics on structured data**  
✅ **Use Cloud Storage for storing raw data, backups, images, or large files**  
✅ **For Data Lakes:** Store raw data in Cloud Storage → Load into BigQuery for analytics

🚀 **BigQuery = Analytics** | **Cloud Storage = File Storage**

Let me know if you need help with setup! 🎯

**You said:**

What is the difference between bigtable and cloud sql

**ChatGPT said:**

**Bigtable vs Cloud SQL: Key Differences & Use Cases**

Both **Cloud Bigtable** and **Cloud SQL** are **managed database services** on **Google Cloud**, but they serve different purposes.

| **Feature** | **Cloud Bigtable (NoSQL)** | **Cloud SQL (Relational SQL DB)** |
| --- | --- | --- |
| **Database Type** | **NoSQL (Wide-column)** | **SQL (Relational, RDBMS)** |
| **Use Case** | Real-time analytics, IoT, time-series data | Traditional applications, OLTP, structured data |
| **Schema** | Schema-less (Column-Family Model) | Fixed Schema (Tables, Rows, Columns) |
| **Query Language** | No SQL (Uses API calls, not SQL) | Uses SQL (MySQL, PostgreSQL, SQL Server) |
| **Scalability** | Horizontally scalable (Petabytes of data) | Vertically scalable (Limited scaling) |
| **Best For** | High-throughput apps (millions of requests/sec) | Transactional applications (banking, e-commerce) |
| **Consistency** | Eventually consistent | Strong consistency |

**What is Cloud Bigtable?**

🔹 **Fully managed NoSQL database** for **high-performance, large-scale applications**  
🔹 Optimized for **real-time analytics, IoT, and time-series data**  
🔹 Uses **HBase API** (Google’s version of Apache HBase)  
🔹 **Supports petabytes of data** and **millions of requests per second**

**Cloud Bigtable Use Cases**

✅ **Time-series Data** (Financial transactions, sensor logs, IoT data)  
✅ **Real-time Analytics** (AdTech, personalization, fraud detection)  
✅ **Machine Learning** (Storing and analyzing ML data)

📌 **Example: Writing Data to Bigtable (Python)**

python

CopyEdit

from google.cloud import bigtable

client = bigtable.Client(project="your-project-id")

instance = client.instance("your-instance-id")

table = instance.table("your-table")

row\_key = b"sensor-123"

row = table.direct\_row(row\_key)

row.set\_cell("data\_family", "temperature", b"25.5")

row.commit()

**What is Cloud SQL?**

🔹 **Managed relational database service** (Supports **MySQL, PostgreSQL, SQL Server**)  
🔹 Best for **OLTP (Online Transaction Processing) applications**  
🔹 **Automatic backups, high availability, failover support**  
🔹 **Integrates easily** with BI tools like **Looker & Power BI**

**Cloud SQL Use Cases**

✅ **E-commerce & Banking Apps** (Transactions, user data storage)  
✅ **ERP & CRM Systems** (Business applications with structured data)  
✅ **Data Warehousing** (Combining with BigQuery for analytics)

📌 **Example: Connecting to Cloud SQL (MySQL)**

sql

CopyEdit

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100) UNIQUE NOT NULL

);

**When to Use Bigtable vs Cloud SQL?**

✅ **Use Cloud Bigtable if you need:**  
✔️ **High-throughput, real-time analytics (millions of reads/writes per second)**  
✔️ **Time-series, IoT, or event-driven data**  
✔️ **Horizontal scalability (Petabyte-scale NoSQL storage)**

✅ **Use Cloud SQL if you need:**  
✔️ **A traditional relational database (MySQL, PostgreSQL, SQL Server)**  
✔️ **Transactional consistency (ACID Compliance)**  
✔️ **Structured, schema-based data for applications**

🚀 **Bigtable = Large-scale, NoSQL, Real-time Analytics**  
🚀 **Cloud SQL = Small-Medium Relational Databases for Applications**

Let me know if you need help setting up either! 🎯

**You said:**

What is Pub/Sub and Cloud Function

**ChatGPT said:**

**Pub/Sub vs Cloud Functions: Key Differences & Use Cases**

Both **Pub/Sub** and **Cloud Functions** are Google Cloud services for event-driven architectures, but they serve different purposes.

| **Feature** | **Pub/Sub (Messaging Service)** | **Cloud Functions (Serverless Compute)** |
| --- | --- | --- |
| **Purpose** | Asynchronous messaging between services | Execute code in response to events |
| **Type** | **Message Broker (Event-Driven System)** | **Event-Driven Serverless Compute** |
| **Use Case** | Decoupling microservices, streaming data | Automating tasks, processing Pub/Sub events |
| **Trigger** | Messages published to topics | Events (Pub/Sub, HTTP, Storage, etc.) |
| **Scalability** | Handles **millions of messages per second** | Scales automatically on demand |
| **Best For** | **Data pipelines, logging, real-time notifications** | **ETL, API processing, automation, lightweight functions** |

**What is Pub/Sub?**

🔹 **Fully managed event-driven messaging service**  
🔹 **Asynchronous communication** between systems  
🔹 Supports **publish-subscribe** pattern  
🔹 **Reliable delivery** (at-least-once, exactly-once ordering)  
🔹 Handles **millions of messages per second**

**Pub/Sub Use Cases**

✅ **Decoupling microservices** (Send & receive messages asynchronously)  
✅ **Streaming analytics** (Real-time data processing in BigQuery, Dataflow)  
✅ **Event-driven workflows** (Trigger Cloud Functions, Cloud Run)

📌 **Example: Publish a Message to a Pub/Sub Topic (Python)**

python

CopyEdit

from google.cloud import pubsub\_v1

project\_id = "your-project-id"

topic\_id = "your-topic"

publisher = pubsub\_v1.PublisherClient()

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

message = "Hello, Pub/Sub!".encode("utf-8")

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

print(f"Message published: {future.result()}")

**What is Cloud Functions?**

🔹 **Serverless compute service** (Runs code without managing servers)  
🔹 **Triggers from events** (Pub/Sub, Cloud Storage, HTTP, Firestore, etc.)  
🔹 **Pay-per-use pricing** (Only runs when triggered)  
🔹 **Supports Python, Node.js, Go, Java, .NET, etc.**

**Cloud Functions Use Cases**

✅ **Automating tasks** (Resize images, process files, send notifications)  
✅ **ETL (Extract, Transform, Load) Pipelines** (Trigger functions on new data)  
✅ **API Processing** (Handle webhooks, process HTTP requests)

📌 **Example: Cloud Function Triggered by Pub/Sub (Python)**

python

CopyEdit

from google.cloud import pubsub\_v1

def pubsub\_trigger(event, context):

message = event['data']

print(f"Received message: {message.decode('utf-8')}")

**When to Use Pub/Sub vs Cloud Functions?**

✅ **Use Pub/Sub if you need:**  
✔️ **Real-time messaging** between services  
✔️ **Scalable event-driven architecture**  
✔️ **Decoupling of microservices**

✅ **Use Cloud Functions if you need:**  
✔️ **To execute code in response to events**  
✔️ **Serverless automation & processing**  
✔️ **ETL, webhooks, API endpoints**

🚀 **Pub/Sub = Event Messaging** | **Cloud Functions = Serverless Event Processing**