**Step 1: Set Up Google Cloud Project**

1. **Create a Google Cloud Project**:
   * Go to the Google Cloud Console.
   * Click the **project dropdown** in the top-right corner and select **New Project**.
   * Name your project, enable billing, and click **Create**.
2. **Enable Required APIs**:
   * **Cloud Functions API**
   * **Cloud Pub/Sub API**
   * **Google Cloud Storage API**
   * **Cloud Scheduler API** (for scheduling tasks)

To enable these APIs:

* + In **Google Cloud Console**, navigate to **APIs & Services** > **Library**.
  + Search for each API and click **Enable** for each one.

**Step 2: Set Up Google Cloud Storage (GCS) Bucket**

1. **Create a GCS Bucket**:
   * In **Google Cloud Console**, go to **Storage** > **Browser**.
   * Click **Create Bucket**.
   * Give your bucket a **unique name** (e.g., my-stock-data-bucket).
   * Choose a **region** and click **Create**.
2. **Set Bucket Permissions**:  
   Ensure your Cloud Function has the correct permissions to write to this bucket.
   * Go to **IAM & Admin** in the console and assign the **Storage Object Admin** role to your Cloud Function's service account.

**Step 3: Create a Pub/Sub Topic**

1. **Create Pub/Sub Topic**:
   * Go to **Google Cloud Console** > **Pub/Sub** > **Topics**.
   * Click **Create Topic**.
   * Give your topic a name (e.g., fetch-stock-data-topic).
   * Click **Create**.

Alternatively, use the following gcloud command to create a Pub/Sub topic:

gcloud pubsub topics create fetch-stock-data-topic

**Step 4: Write the Cloud Function Code**

You will create a Cloud Function that is triggered by **Pub/Sub**, fetches data from an API, and uploads it to **Google Cloud Storage (GCS)**.

**4.1 Create Function Code (main.py):**

1. **Fetch data from the API** (Alpha Vantage in this case).
2. **Upload the raw data to Google Cloud Storage**.

**4.2 Create a requirements.txt file:**

This file lists the necessary dependencies (libraries) for your Cloud Function.

**requirements.txt**:

requests

google-cloud-storage

**Step 5: Deploy the Cloud Function**

You need to deploy your Cloud Function and link it to the **Pub/Sub topic**.

1. **Deploy Cloud Function**:  
   Use the gcloud command to deploy the function:

gcloud functions deploy ingest\_data\_from\_pubsub \

--runtime python310 \

--trigger-topic fetch-stock-data-topic \

--allow-unauthenticated

* **--trigger-topic fetch-stock-data-topic**: This links the Cloud Function to the **Pub/Sub topic** (fetch-stock-data-topic).
* **--runtime python310**: Specifies the Python runtime (Python 3.10).
* **--allow-unauthenticated**: Allows unauthenticated access to the function (useful for testing).

**Step 6: Set Up Cloud Scheduler to Trigger Every 5 Minutes**

Now that the **Cloud Function** is deployed and linked to the **Pub/Sub topic**, you need to create a **Cloud Scheduler job** to publish a message to the **Pub/Sub topic** every 5 minutes.

1. **Create Cloud Scheduler Job**:

Use the following command to create a **Cloud Scheduler job** that publishes a message to the **Pub/Sub topic** every 5 minutes:

bash

Copy

gcloud scheduler jobs create pubsub fetch-stock-data-job \

--schedule "\*/5 \* \* \* \*" \ # Cron expression for every 5 minutes

--topic fetch-stock-data-topic \

--message-body "trigger"

* **--schedule "\*/5 \* \* \* \*"**: This cron expression means "every 5 minutes".
* **--topic fetch-stock-data-topic**: This is the **Pub/Sub topic** to which the message will be published.
* **--message-body "trigger"**: The message that will trigger the Cloud Function.

**Step 7: Test the System**

1. **Manually Trigger the Pub/Sub Message**:  
   You can manually trigger the **Cloud Function** by publishing a message to the **Pub/Sub topic**:

bash

Copy

gcloud pubsub topics publish fetch-stock-data-topic --message "trigger"

This will publish a message to the topic, which will trigger the **Cloud Function** to fetch data and upload it to **GCS**.

1. **Check Logs**:  
   Go to **Google Cloud Console** > **Cloud Functions** > Your function, and check the **Logs** to see if the function was triggered successfully.
2. **Check Google Cloud Storage**:  
   Go to **Google Cloud Console** > **Storage** > **Browser**, and verify that the file raw\_data/ibm\_stock\_data.json has been uploaded.
3. **Monitor the Scheduled Trigger**:  
   After setting up the **Cloud Scheduler**, you can verify the logs for every 5-minute interval to make sure the function is being triggered automatically.

**Step 8: Data Transformation Process**

After the raw data is ingested into the **raw\_data/** folder in **GCS**, you will need to transform this data before loading it into **BigQuery**. This typically involves:

1. **Fetching the Raw Data** from **GCS**.
2. **Transforming the Data** (e.g., calculating the **5-day moving average**).
3. **Saving the Transformed Data** to a **new folder in GCS** (transformed\_data/).
4. **Deleting Previous Files** from **GCS** to ensure only the latest file remains.

**Step 9: Create Cloud Function to Transform and Upload Data**

You can create another **Cloud Function** that is triggered by new files uploaded to the **raw\_data/** folder, perform the transformation (e.g., calculate the moving average), and save the transformed data in **transformed\_data/**.

**Step 10: Deploy the Transformation Cloud Function**

1. **Create a requirements.txt** file for the Cloud Function:

google-cloud-storage==1.43.0

pandas==1.3.3

1. **Deploy the Cloud Function**:

Use the following command to deploy your **Cloud Function** that handles the transformation process:

bash

Copy

gcloud functions deploy transform\_and\_upload \

--runtime python310 \

--trigger-resource your-bucket-name \

--trigger-event google.storage.object.finalize \

--allow-unauthenticated

* + **--trigger-resource**: The GCS bucket where the raw data is stored.
  + **--trigger-event google.storage.object.finalize**: This triggers the function when a new file is uploaded to **GCS**.
  + **--allow-unauthenticated**: Allows unauthenticated access (for testing, adjust for production).

**Step 11: Load Transformed Data into BigQuery**

After the **transformed data** is saved in **GCS** under the **transformed\_data/** folder, you can load it into **BigQuery**.

1. **Create a BigQuery Dataset and Table**:
   * In **Google Cloud Console**, go to **BigQuery** > **Create Dataset** and create a dataset (e.g., stock\_market\_data).
   * Create a table in this dataset (e.g., transformed\_stock\_data).
2. **Load Transformed Data into BigQuery**:  
   You can load the **transformed data** from **GCS** into **BigQuery** using **Python**:

from google.cloud import bigquery

def load\_to\_bigquery():

# Set up BigQuery client

bq\_client = bigquery.Client()

# Define GCS path and BigQuery table info

gcs\_uri = "gs://your-bucket-name/transformed\_data/output.csv"

dataset\_id = "your-project-id.your-dataset-id"

table\_id = "your-project-id.your-dataset-id.transformed\_stock\_data"

# Set the job configuration for loading the CSV file

job\_config = bigquery.LoadJobConfig(

source\_format=bigquery.SourceFormat.CSV,

skip\_leading\_rows=1,

autodetect=True,

)

# Load data from GCS into BigQuery

load\_job = bq\_client.load\_table\_from\_uri(

gcs\_uri, table\_id, job\_config=job\_config

)

# Wait for the job to complete

load\_job.result()

print(f"Loaded data from {gcs\_uri} to BigQuery table {table\_id}")

# Run the function

load\_to\_bigquery()

1. **Verify the Load**:  
   After loading the data, you can query **BigQuery** to ensure the data has been loaded:

SELECT \* FROM `your-project-id.your-dataset-id.transformed\_stock\_data` LIMIT 10;

**Step 12: Delete Files in the transformed\_data/ Folder**

Once the **data is loaded into BigQuery**, you can delete **old transformed files** in **GCS** to avoid storage costs. You can automate this deletion process in the **Cloud Function** after the data is loaded.

**Python Code to Delete Files in GCS:**

from google.cloud import storage

def delete\_old\_files(bucket\_name):

"""Delete all files in the transformed\_data folder in GCS"""

storage\_client = storage.Client()

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

# List the files in the transformed\_data folder

blobs = bucket.list\_blobs(prefix="transformed\_data/")

# Delete each file

for blob in blobs:

blob.delete()

print("Previous transformed files deleted.")

# Call the function to delete old files

delete\_old\_files("your-bucket-name")

**Step 13: Automate the Entire Process**

To **automate the entire pipeline**, you need to:

1. **Ingest raw data** into **GCS** (this can be done manually or through the first **Cloud Function**).
2. **Trigger the transformation function** automatically when new raw data is uploaded.
3. **Load the transformed data into BigQuery**.
4. **Delete the previous files** in **GCS** after successful data processing.

**Step 14: Connect BigQuery to Looker for Visualization**

1. **Set Up BigQuery in Looker**:
   * Go to **Looker** > **Admin** > **Connections**.
   * Create a **connection** for **BigQuery** by providing your **Google Cloud Project ID** and authentication details (either **OAuth** or **Service Account**).
2. **Create LookML Models**:  
   In Looker, create **LookML models** and **views** based on the BigQuery tables.
3. **Create Explores and Dashboards**:
   * Use **Looker Explore** to create visualizations like **line charts**, **bar charts**, etc.
   * Combine multiple **Looks** into **Dashboards** to visualize your stock data (e.g., stock price trends, moving averages).