**DATA LOADING USING CLOUD FUNCTIONS FROM GOOGLE CLOUD STORAGE TO BIGQUERY**

A Project Report Submitted in the fulfilment of the requirements for Final-Project Evaluation

**SUBMITTED BY:**

**NAME: VAISHNAVI R**

**EMP ID: 2319882**

**CSDAIA24GP002**

**A white background with black dots

Description automatically generated**

**ABSTRACT**

This project aims to automate the process of loading data from Google Cloud Storage (GCS) to Big Query using Google Cloud Platform (GCP) services. The pipeline involves creating a GCS bucket to store data files, setting up a Cloud Function to trigger upon file creation events in the bucket, and developing the function to load data from the newly created files into Big Query tables. Additionally, a Big Query table will be created to store the incoming data.

**Steps:**

1. **Create a Bucket:**

Utilize the Google Cloud Console or Cloud SDK to create a new GCS bucket. Configure bucket settings such as access controls and storage class according to project requirements.

1. **Develop a Cloud Function:**

Define a Cloud Function that will be triggered whenever a new file is added to the GCS bucket. Set up the necessary permissions for the Cloud Function to access the GCS bucket. Implement logic within the Cloud Function to handle file creation events and trigger data loading into Big Query.

1. **Data Loading to Big Query:**

Integrate the Cloud Function with Big Query to load data from the newly created files. Utilize Google Cloud Client Libraries or other suitable methods to interact with Big Query from the Cloud Function. Implement error handling and data validation procedures within the Cloud Function to ensure data integrity during the loading process.

1. **Create Table in BigQuery:**

Define the schema for the BigQuery table to match the structure of the incoming data files. Use Google Cloud Console, BigQuery API, or CLI commands to create the table in the designated dataset. Configure table properties such as partitioning and clustering for optimal query performance if required.

By completing the above tasks, the project will establish an end-to-end automated data loading pipeline from Google Cloud Storage to BigQuery. This pipeline will enable efficient and reliable processing of data, ensuring timely availability of insights for analysis and decision-making purposes.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **S.NO** | **CONTENT** |
| **1** | **INTRODUCTION** |
| **2** | **COMPONENTS OF GCP** |
| **3** | **SOFTWARE REQUIREMENTS** |
| **4** | **PROJECT STATEMENT** |
| **5** | **PROJECT REQUIREMENTS** |
| **6** | **IMPLEMENTATION** |
| **7** | **CONCLUSION** |

**INTRODUCTION**

In the realm of modern data management, organizations seek efficient and scalable solutions for ingesting, processing, and analyzing vast volumes of data. This project embarks on building an end-to-end data integration pipeline within Google Cloud Platform (GCP), focusing on seamlessly transferring data from Google Cloud Storage (GCS) to BigQuery. By harnessing the capabilities of GCP services, including Cloud Storage, Cloud Functions, and BigQuery, this project aims to automate the data ingestion process, enabling rapid insights extraction and informed decision-making.

**Key Objectives:**

**1. Create a Bucket:**

   The project initiates by establishing a GCS bucket, providing a centralized repository for storing incoming data files. This bucket serves as the foundation for the data ingestion pipeline, offering scalability, durability, and accessibility for managing diverse data sources.

**2. Develop a Cloud Function:**

   Subsequently, a Cloud Function will be developed to trigger automatically upon file creation events within the designated GCS bucket. Leveraging the event-driven architecture of Cloud Functions, this component ensures real-time responsiveness to incoming data, initiating data processing workflows without manual intervention.

**3. Data Loading to BigQuery:**

   The core functionality of the Cloud Function revolves around extracting data from the newly created files within the GCS bucket and loading it into BigQuery. This step involves implementing robust data loading mechanisms to facilitate seamless and efficient transfer of data, preserving its integrity and structure during the ingestion process.

**4. Create Table in BigQuery:**

   As part of the pipeline, tables will be created within BigQuery to accommodate the incoming data. These tables are designed to align with the schema of the incoming data files, ensuring compatibility and optimized querying performance for subsequent analytics and reporting tasks.

By orchestrating the above components within the Google Cloud Platform, this project endeavors to establish a resilient and automated data integration pipeline, seamlessly bridging Google Cloud Storage and BigQuery. Through efficient data ingestion, transformation, and storage, organizations can unlock the full potential of their data assets, empowering data-driven decision-making and fostering innovation in various domains.

**COMPONENTS OF GCP**

Google Cloud Platform (GCP) offers a wide range of services and products to support various cloud computing needs. Here are some key components of GCP:

**1. Compute:** Services for virtual machines (Compute Engine), managed Kubernetes clusters (Google Kubernetes Engine), and serverless computing (Cloud Functions, App Engine).

**2. Storage:** Scalable and durable object storage (Cloud Storage), relational and non-relational databases (Cloud SQL, Firestore, Bigtable), and file storage (Filestore).

**3. Networking:** Virtual private cloud (VPC), global load balancing, content delivery network (CDN), and network monitoring and security.

**4. Big Data:** Managed data warehousing (BigQuery), real-time data analytics (Dataflow, Dataprep), big data processing (Dataproc), and machine learning (AI Platform, AutoML).

**5. Identity and Security:** Identity and Access Management (IAM), key management (Cloud KMS), security monitoring (Cloud Security Command Center), and data loss prevention (DLP).

**6. Management Tools:** Monitoring, logging, and diagnostics (Stackdriver), deployment and automation (Deployment Manager, Cloud Build), and resource optimization (Cost Management, Cloud Scheduler).

**7. Developer Tools:** Software development kits (SDKs), integrated development environments (IDEs), continuous integration and continuous deployment (CI/CD) tools, and APIs.

**8. AI and Machine Learning:** Pre-trained models (Vision AI, Natural Language AI), custom model training (AI Platform), and specialized hardware accelerators (TPUs).

These components provide a foundation for building and managing applications, running workloads, storing and analyzing data, and leveraging artificial intelligence and machine learning capabilities within the Google Cloud Platform ecosystem.

**SOFTWARE REQUIREMENTS**

For the project involving the creation of a data integration pipeline from Google Cloud Storage (GCS) to BigQuery, the following software requirements are necessary:

**1. Google Cloud Platform Account:**

   - Sign up for a Google Cloud Platform (GCP) account to access the required services and resources.

**2. Google Cloud Storage:**

   - Utilize Google Cloud Storage (GCS) to create a bucket for storing incoming data files.

   - Access and manage GCS resources through the Cloud Console or SDK.

**3. Google Cloud Functions:**

   - Develop Cloud Functions to trigger upon file creation events within the GCS bucket.

   - Configure and deploy Cloud Functions using the Cloud Console or SDK.

   - Write code for the Cloud Function in a supported programming language such as Node.js, Python, or Go.

**4. Google BigQuery:**

   - Leverage Google BigQuery for storing and querying large datasets.

   - Define schema and create tables within BigQuery to accommodate the incoming data.

   - Access and manage BigQuery resources through the Cloud Console or SDK.

**5. Authentication and Permissions:**

   - Configure authentication and access permissions to enable communication between services.

   - Utilize service accounts and IAM roles to grant necessary permissions for accessing GCS and BigQuery resources.

**6. Programming Language and Libraries:**

   - Depending on the chosen programming language for developing Cloud Functions, ensure compatibility with GCP SDKs and client libraries.

   - Install any required dependencies or libraries for interacting with GCS and BigQuery services within the Cloud Function code.

**7. Development Environment:**

   - Set up a suitable development environment for writing, testing, and deploying Cloud Functions. Choose an integrated development environment (IDE) or code editor that supports the selected programming language.

**PROJECT STATEMENT**

The objective of this project is to develop a data loading using Google Cloud Platform (GCP) services, specifically targeting the seamless transfer of data from Google Cloud Storage (GCS) to BigQuery. The pipeline will involve creating a GCS bucket to store incoming data files, setting up a Cloud Function to trigger upon file creation events within the bucket, developing the Cloud Function to load data from the files into BigQuery tables, and creating corresponding tables in BigQuery to accommodate the incoming data.

The project will focus on implementing the necessary components within GCP to establish a robust and automated data ingestion pipeline. It will cover the creation of resources such as GCS buckets and BigQuery tables, as well as the development and deployment of Cloud Functions to orchestrate data transfer and loading processes. The project will primarily target text-based data files for simplicity, but the pipeline architecture will be designed to accommodate various file formats and data types.

**PROJECT REQUIREMENTS**

**1. Google Cloud Platform Account:**

   - Access to Google Cloud Platform (GCP) services is required to create and manage resources such as buckets, Cloud Functions, and BigQuery.

**2. Google Cloud Storage:**

   - Create a GCS bucket to store incoming data files.

   - Specify the bucket name and choose the appropriate storage class based on data retention and access requirements.

**3. Cloud Functions:**

   - Develop a Cloud Function that triggers upon file creation events in the GCS bucket.

   - Choose the appropriate programming language (e.g., Node.js, Python, Go) supported by Cloud Functions.

   - Define the trigger event type (e.g., `google.storage.object.finalize`) and specify the GCS bucket to monitor.

**4. Data Loading to BigQuery:**

   - Configure the Cloud Function to load data from the newly created files into BigQuery.

   - Use the BigQuery client library within the Cloud Function code to interact with BigQuery.

   - Ensure proper error handling and data validation mechanisms are implemented to maintain data integrity during the loading process.

**5. Google BigQuery:**

   - Define the schema for the BigQuery table(s) to accommodate the incoming data.

   - Create the table(s) within BigQuery using SQL DDL (Data Definition Language) statements or via programmatic methods.

   - Specify table properties such as partitioning and clustering for optimized query performance if required.

**6. Authentication and Permissions:**

   - Configure authentication and access permissions to allow the Cloud Function to interact with GCS and BigQuery.

   - Use service accounts and IAM roles to grant the necessary permissions for reading from GCS, writing to BigQuery, and managing resources.

**7. Documentation and Deployment:**

   - Document the configuration steps, code implementation, and deployment process for future reference.

   - Deploy the Cloud Function and associated resources to the GCP environment.

By fulfilling these project requirements, you can successfully implement an automated data integration pipeline from Google Cloud Storage to BigQuery, enabling efficient data transfer and analysis within the GCP ecosystem.

**IMPLEMENTATION**

**CODE:**

import os

from google.cloud import storage, bigquery

def move\_file(data, context):

"""Moves files between buckets or loads CSV files into BigQuery."""

source\_bucket = data['bucket']

source\_file = data['name']

file\_extension = os.path.splitext(source\_file)[1].lower()

if file\_extension == '.csv':

load\_csv\_into\_bigquery(source\_bucket, source\_file)

def load\_csv\_into\_bigquery(bucket\_name, file\_name):

"""Loads CSV file into BigQuery."""

bq\_client = bigquery.Client()

dataset\_id = 'demo\_dataset\_0102'

table\_id = os.path.splitext(file\_name)[0]

bucket\_path = f"gs://{bucket\_name}/{file\_name}"

table\_ref = bq\_client.dataset(dataset\_id).table(table\_id)

job\_config = bigquery.LoadJobConfig(

autodetect=True,

source\_format=bigquery.SourceFormat.CSV,

skip\_leading\_rows=1,

)

load\_job = bq\_client.load\_table\_from\_uri(bucket\_path, table\_ref, job\_config=job\_config)

load\_job.result() # Wait for the job to complete

# Entry point for Cloud Function

def main(event, context):

move\_file(event, context)

**STEP 1: Giving the IAM permissions**

**A screenshot of a computer

Description automatically generated**

**STEP 2: Creating a VM Instance:**

**A screenshot of a computer

Description automatically generated**

**logging in to gcloud using VM Instance:**

**A screenshot of a computer

Description automatically generated**

**STEP 2: Create a Bucket:**

**Using VM Instance:**



**Using Console:**

**A screenshot of a computer

Description automatically generated**

**STEP 3: Create a dataset in Bigquery:**

**Using VM Instance:**

****

**Using Console:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated**Dataset created Successfully:**

A screenshot of a computer

Description automatically generated**STEP 4: Create a function.**

**Code to trigger cloud function to bigquery:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated**Function deployed with the following metrics:**

**Details of the cloud function:**

**A screenshot of a computer

Description automatically generated**

**Trigger type of the cloud function:**

**A screenshot of a computer

Description automatically generated**

**STEP 5: Load file into the bucket**

A screenshot of a computer

Description automatically generated

**Preview of the table:**

**A screenshot of a computer

Description automatically generated**

**Logs:**

**A screenshot of a computer

Description automatically generated**

**We have loaded the file into the bigquery – Bigquery table**

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer

Description automatically generated**

**Uploading more Files to the Bucket:**

**A screenshot of a computer

Description automatically generated**

**Logs:**

**A screenshot of a computer

Description automatically generated**

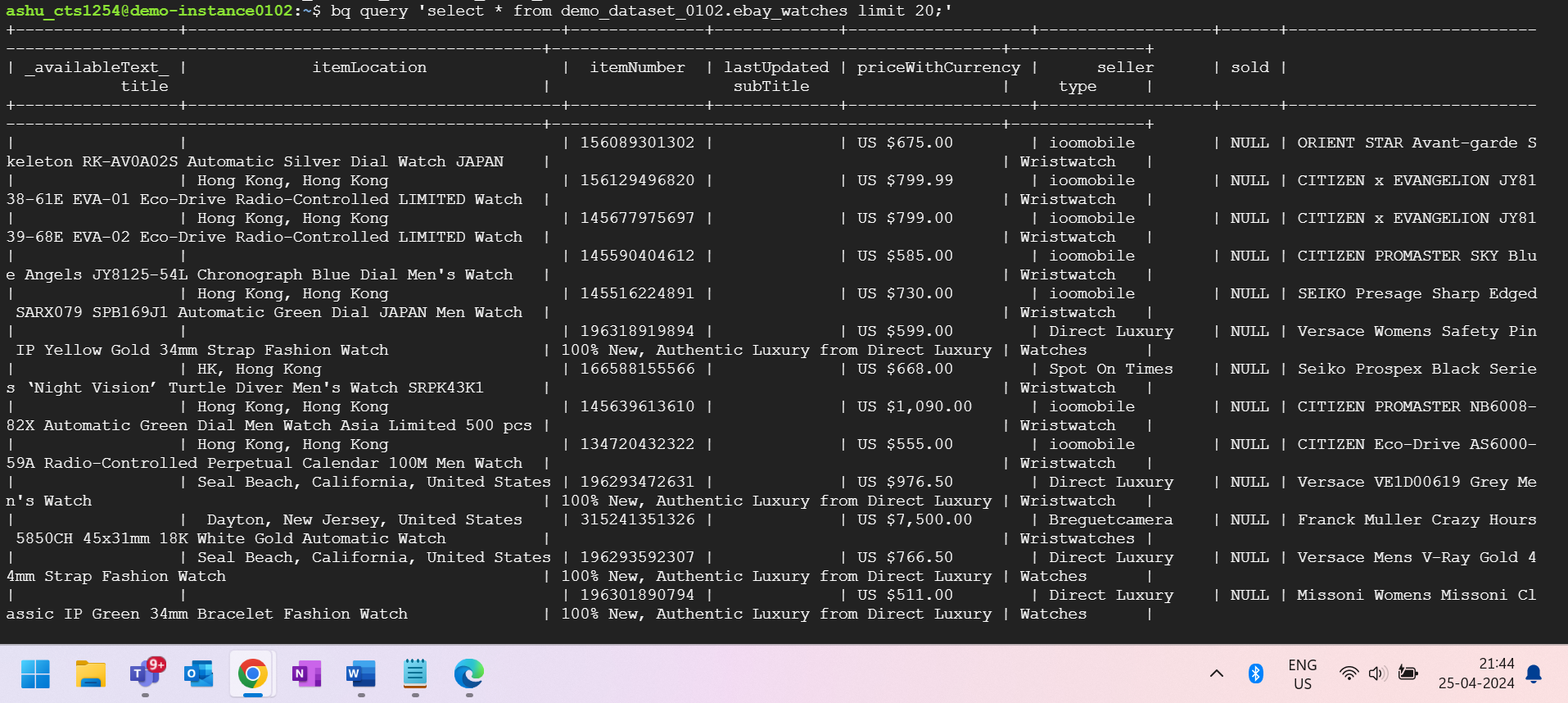
**New Table is Created in BigQuery Dataset:**

****

**A screenshot of a computer

Description automatically generated**

**Query:**

****

**CONCLUSION**

In conclusion, the successful implementation of data loading from Google Cloud Storage (GCS) to BigQuery marks a significant milestone in enhancing data management and analytics capabilities within the Google Cloud Platform (GCP) ecosystem. By completing the key objectives of creating a bucket, developing a Cloud Function triggered by file creation events, loading data from files into BigQuery, and creating tables within BigQuery, the project has achieved its primary goal of establishing a seamless and efficient data ingestion pipeline.

Through the creation of a GCS bucket, organizations gain a centralized repository for storing incoming data files, providing scalability, durability, and accessibility for managing diverse data sources. The development of a Cloud Function enables real-time responsiveness to file creation events within the bucket, initiating automated data loading processes into BigQuery. Leveraging BigQuery’s powerful analytics capabilities, the Cloud Function facilitates the transformation and loading of data from files into designated tables, enabling timely insights extraction and decision-making.

Furthermore, the creation of tables within BigQuery ensures proper organization and structuring of the incoming data, optimizing querying performance and facilitating downstream analytics workflows. By defining table schemas and properties, the project lays the foundation for efficient data analysis, reporting, and visualization within BigQuery, empowering organizations to derive actionable insights from their data assets.

Overall, the automated data integration pipeline enhances data agility, reliability, and scalability within the GCP environment, enabling organizations to streamline data management processes and unlock the full potential of their data for driving innovation and strategic decision-making. As data volumes continue to grow, the project provides a robust framework for effectively managing data flows, enabling organizations to stay competitive in today's data-driven landscape.