Hy Pham

BigQuery Automation Exports via Email

Metropolia University of Applied Sciences

Master of Engineering

Information Technology

Master's Thesis

05 December 2022

PREFACE

Finding a topic for thesis that is both relevant and interesting may be challenging. That statement is accurate. In fact, I am now refocusing my thesis for the third time. I'm searching for a solution to solve real-world problems while still benefiting customers; if I could only locate the appropriate product, I may find it. While working on the internal project and writing my thesis at the same time, I learned a great deal about the Google Cloud Platform and infrastructure as code for automation. I also came to the conclusion that writing my thesis was the best way for me to keep track of and think about the work and research I had done for the project.

I would like to say thanks to Professor Ville Jaaskelainen and Professor Sami Sainio for all the advice, feedback, and support during my studies.

My partner and my cousin have been a fantastic source of support and inspiration for me as I try to complete my thesis, especially with regards to my studies in Metropolia, and I would want to express my appreciation to them both.

Espoo, 05 December 2022

Hy Pham

Abstract

Author:	Hy Pham
Title:	BigQuery Automation Exports via Email
Number of Pages:	54 pages + 6 appendices
Date:	05 December 2022
Degree:	Master of Engineering
Degree Programme:	Information Technology
Professional Major:	Networking and Services
Supervisor:	Sami Sainio, Principal Lecturer

Many clients of Google Cloud choose to export their billing data which contains price data, usage and cost estimates to CSV or JSON files straight from the supported billing export tool. This allows them to do future monitoring and forecasting. Google Cloud Platform (GCP) offered this capability to its clients as default; however, Google no longer provides this service.

This thesis offers guideline how to implement an alternative solution, which will be presented as a stand-alone Terraform module in the GCP Landing Zone.

The thesis describes the process of how to leverage other export billing data to BigQuery (BQ) feature. Other popular GCP services such as Cloud Functions, Pub/Sub, Cloud Storage, third-party mail server SendGrid, and Infrastructure as Code Terraform are also discussed. These tools can be used to build the automated billing data export solution for the case company clients as well as others, including the FinOps team, Architect, and DevOps team managers.

Keywords:

Google Cloud Platform, BigQuery, Cloud Functions, Pub/Sub, Cloud Storage, SendGrid, Terraform

Contents

List of Figures

List of Abbreviations

BQ	BigQuery
FinOps	Financial Operations
SQL	Structured Query Language
ID	Identification
SDK	Software Development Kit
GCS	Google Cloud Storage
ETL	Extract, Transform, and Load
APIs	Application Programming Interface
GCP	Google Cloud Platform
IoT	Internet Of Thing
IT	Information Technology
SKU	Stock Keeping Unit
URL	Uniform Resource Locator
DML	Data Manipulation Language

1 Introduction

Monitoring, evaluating, and optimizing expenditures is now an integral part of managing a business's cloud environment, but dealing with all the company's cost and consumption data may be challenging. Managing this data, there are data management tools offered by cloud service providers, such as by Google. Customers can confidently expand their businesses in the cloud because of the enhanced visibility, accountability, control, and insight provided by the cost management solutions. These technologies assist in decreasing the complexity of company’s cloud expenditures and enhance the predictability of those costs. They are also tailored to fit the requirements of enterprises of all sizes.

FinOps is a growing cloud financial management discipline and cultural practice that facilitates collaboration across engineering, finance, technology, and business teams to make expenditure choices based on data. FinOps is an approach wherein end-users are responsible for their own cloud use, its associated expenses, and the impact of that spending on the company's bottom line.

1.1 Case Company

The case company is a European market leader in cloud implementation, application development, managed services, and training. They are also a cloud-native pioneer with a track record of assisting organizations in using the public cloud in a manner that combines rapid wins, instant savings, and long-term value.

The case company works with VMware and is certified by Microsoft Azure, Google Cloud Platform, and Amazon Web Services. It also takes part in the Gartner Magic Quadrant. [1]

The case company has ten European centres, over 1700 workers, and has completed over 1,000 successful cloud projects for mid-size to big corporations.

1.2 Billing data

Billing data is quite granular and can be as detailed as the expense of a single request, supplied kilobyte, or resource utilization second. This granularity fulfills the ultimate promise of pay-as-you-go. On the other hand, when a large number of public cloud services are used, the billing data becomes massive and eventually impossible to understand. Clarity on the costs of the public cloud necessitates an in-depth understanding of what is generating these expenses, how they may be contained, and finally, who should bear these costs. If expenditures are incorrectly allocated inside an organization, a pay-as-you-go strategy will not be effective. In addition, carriers offer a variety of strategies to lower service prices by selecting the appropriate service types, including bulk discounts, reservations, and discounts for continuing use. Purchasing public cloud services has become both more flexible and more complex. Most clients are uncontested leaders in their own industries. They leverage the public cloud to develop new products and services, provide daily value to their consumers, and disrupt their respective sectors. Furthermore, the customers focus on public cloud cost monitoring, monthly budgets, and cost-cutting strategies. The mission of the case company is to support customers with cost optimization and cost clarity. Customers can monitor and track their cloud expenditures based on the case company’s Insight tools and services in multiple ways. They also offer cost-optimization recommendations for the public cloud that are supported by both artificial and actual intelligence. There are two ways in which the case company has proven successful in assisting businesses in eliminating waste and making the most of their cloud resource budgets. Using the right-sizing and pricing models, Cost Consulting helps to improve production environments, while Cost Cleanup improves testing environments by eliminating unnecessary waste.

1.3 Objective

This project's objective is to provide an alternative solution for clients and internal teams, such as the FinOps team and the Architect team, to quickly receive billing reports regarding cloud usage expenses via email without having to log in to the Google Cloud Platform. In this project, the usefulness of data is reliant on its analysis and accessibility. It is possible to export query results from BigQuery as scheduled emails. This sends an email to end users including a link to the most recent query results, which is excellent for daily business process statistics, monthly website metrics summaries, and weekly business assessments. Through email, stakeholders may quickly get pertinent information for each inquiry.

2 Theoretical Background

This chapter describes the necessary theoretical context for understanding and implementing the recommended solution.

2.1 BigQuery (BQ)

BQ [2], a corporate data warehouse built on Google's infrastructure, is designed to ingest, store, analyze, and visualize data with ease. It is also Google’s serverless service, which means users only need to concentrate on their data rather than managing the resources. Users can set permissions for others to view the project or run data queries based on their specific business needs. BQ can be accessed through the BQ command line tool, cloud console, and REST API requests through BQ client libraries such as Python, Go, NodeJS, and other languages. Moreover, users can utilize many third-party utilities to access BQ, such as tools for data visualization and data loading. The BQ service is completely managed by GCP. There is no prerequisite deployment of disks or virtual machines to begin operation.

Each project has its own dataset. Datasets are used to organize and manage access to nested database objects like tables and views. To start importing data into BQ, users must first create at least one dataset, which they will use to link any table or view they want to import.

Data from a BQ table is stored in rows. Columns are used to create each individual record. A schema describing the data types, field or column names and others is used to define each table. Users can either provide a table's schema when it is generated, build tables without one, define the schema in BQ query jobs, or load a task that populates the table database with data. There are table formats that are supported by BQ. Native tables are tables supported by BQ's native storage. External tables that are supported by storage outside of BQ such as Cloud Storage, Cloud SQL and Google Drive. Views are logical representations of data in the form of tables, specified by a SQL statement. [2]

The SQL commands execution and user-defined functions costs are referred to query pricing. BQ calculates query fees based on a single metric and the quantity of bytes handled. Users are charged based on the number of bytes processed, regardless of where the data is stored. It can be either in BQ or external data source like GCS and Cloud Bigtable. Monthly processing of the first terabyte of data is totally free each billing account. Users will be charged $5 per TB each month after that. [4]

There are multiple methods for importing data into BQ [5]. Using the batch loading function, the source data can be loaded into a BQ table all at once. An external database, log files, and a CSV file are all potential components of the data source. This category includes ETL tasks. The uploaded data type is free. Streaming involves continuously sending smaller batches of data in real-time so that it can be queried as it arrives. Apply queries to generate data, then insert or replace it in a table. DML commands can be used to create batch updates to the existing database or to store the query results to a new database.

Different file formats have different speeds for ingesting data, and many of them are supported. Avro is row binary file format. It is suitable for maximizing loading speed. Both Optimized Row Columnar (ORC) and Parquet are columnar formats. BQ reads the entire record when consuming data. ORC and Parquet files load slower than Avro because they are columnar formats. Since GZIP (GNU zip) compression cannot be split, each compressed file must be decompressed before the process can be run in parallel. This means that compressed CSV and JSON files perform less well than uncompressed ones. Figure 1 shows the loading speed of the various data formats.

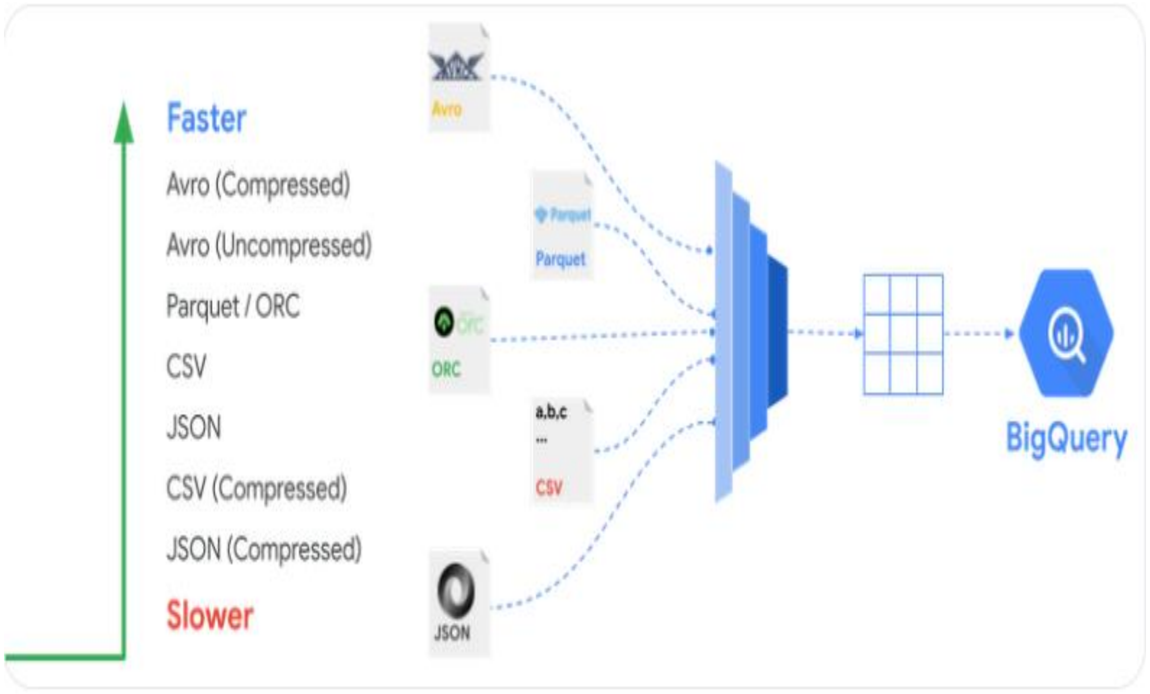


Figure 1: BQ loading different data formats.

All query results are stored in a BQ table. The BQ table is used to keep track of the results of all queries. The table is either one in which the user has chosen to save their data or a cached version of the data. Per-user and per-project temporary tables of cached results are maintained. If query results are written to a permanent table, storage fees are incurred. All query results are stored for roughly 24 hours in temporary tables. In this case, the user will only be charged once for processing the stored data, even if they execute the same query numerous times. [6]

Figure 2 shows the implementation of BQ in real-time analytics. Successfully respond to business events in real-time with the help of event-driven analysis. Streaming data can be imported into BQ and made available for queries because of its built-in streaming features. Therefore, companies can maintain flexibility and base business decisions on the most up-to-date information. On the other hand, a full streaming analytics solution can be made with the help of Dataflow, which makes it easy to set up quick pipelines for streaming data.

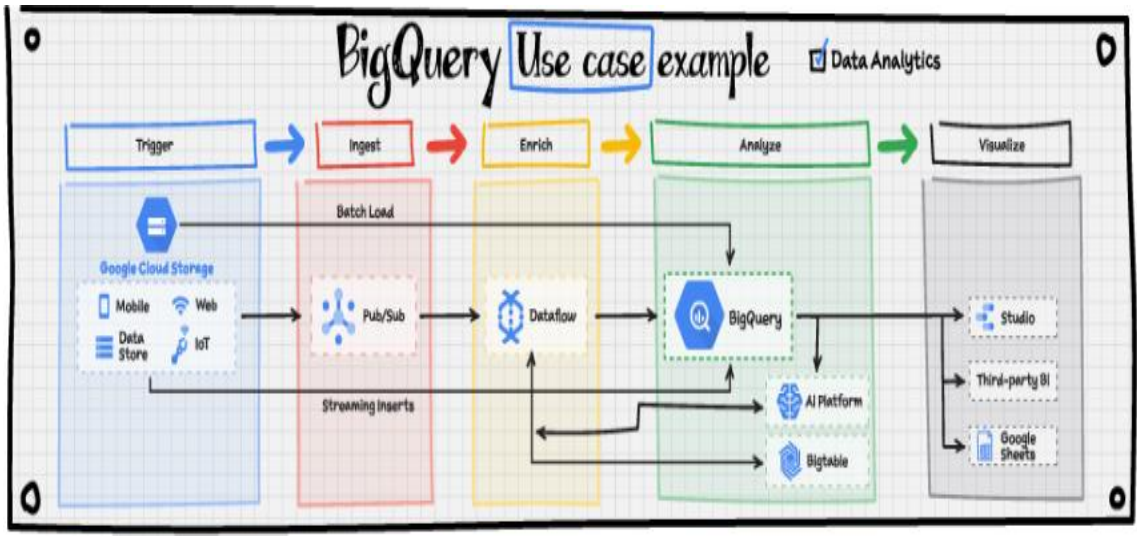


Figure 2: BQ in Data Analytics [7]

2.2 Google Cloud Storage (GCS)

GCS [8] is a Google Cloud Platform objects storage service that store unchangeable data such as images, videos, text and other file types globally, scalability and security. Users can update or retrieve data as frequently as the application requires. The stored objects possess an ID, metadata, attributes, and data. Metadata could consist of a variety of information, such as the file's security classification, the apps that can access it, and similar data. Object storage is a good choice for a wide range of applications, from web serving to data analytics, because of its ID, metadata, and properties.

The following are some methods for utilizing GCS: [9]

- Google Cloud Console: users can manage data by using the console via a web browser.
- gsutil is a command-line utility that enables Cloud Storage interaction via a terminal. Downloading the Cloud Software Development Kit, that comes with gsutil and the gcloud utility for additional services, is an option if users utilize other GCP services.
- To work with data in REST APIs, users can use either the XML or JSON API.
- Google Client Libraries help streamline the data using various programming languages, such as Python, Go, and Node.js to name a few.

Figure 3 shows the example use case of cloud storage classes [10]. Users can choose standard storage class for storing the data that can be accessed frequently and globally, like when streaming movies, doing interactive work, providing data for gaming and mobile applications, or delivering website content. For the data that only needs to be accessed and updated once or less than a month, nearline storage class is the good choice. It can be used for often uploading files to cloud storage, but you can only access them once per month for analysis. Data that is only accessed and changed once quarterly is perfect with coldline storage class option. However, it's important to keep in mind that the most cost-effective storage class is archive which can be used for data which can just being stored for backup or archiving reasons. The archive storage class is the optimal solution for storing data which will be accessed less frequently than annually, such as cold data storage and disaster recovery.

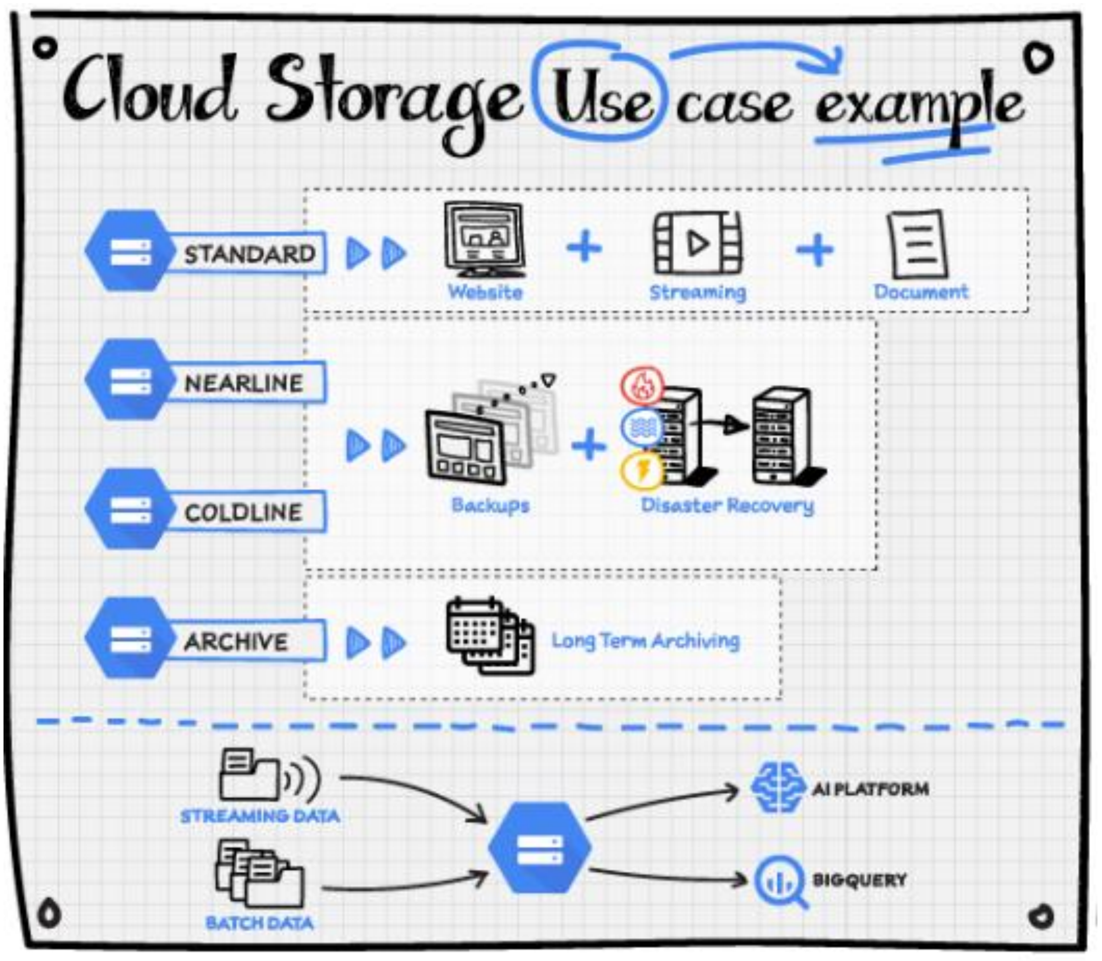


Figure 3: Google Cloud Storage class use cases [11]

Object storage provides a variety of solutions that can assist a company. Persistent data store that is provided by using Google Cloud object storage effectively support the building and migrating to cloud-native applications. Holding numerous amounts of any types of data, big data analytics provided by object storage can prove to have significant support and help gain valuable comprehension of customers, operations, or markets. Additionally, object storage seems to have couples of benefits relating to financial effectiveness. Rich media, including music, video, and images, is effectively stored and distributed where it reduces costs. There might be no need to spend expenses in the backups or archives while immediate access to all data is always possible. Object storage also aims to balance management of machine-to-machine data and advancing analytics to make sense of artificial intelligence.

2.3 Cloud Functions (CF)

When it comes to creating and integrating cloud services, Google CF [12] provides a serverless execution environment that provides a comprehension solution. By using Cloud Functions, users can build small, purpose-built programs that react to certain events in the cloud environment. When a monitored event occurs, the event is triggered. This is a controlled environment in which the code will run. No infrastructure provisioning or server administration is required.

With Google Cloud Platform, users can create Cloud Functions in JScript, Python, Ruby, and C#. The function is both portable and locally tested because it can be run in Node.js 10+, Python 3.7+, Go 1.11+, and Java 11+ environment.

Using Cloud Functions, users will leave administering servers, setting up software, upgrading frameworks, or patching OS systems to the service provider. Google handles the software and infrastructure; users are responsible for the functional code. Moreover, resources are made immediately available in the events. In other words, a function's invocation rate can be scaled automatically from a few per day to millions per day.

Events are occurrences in the cloud infrastructure that prompt users to take some sort of action. Examples include making modifications to a database, adding new files to GCS, or booting up a virtual machine’s new instance. There are multiple events that are supported by CF. Data processing like Extract-Transfer-Load

(ETL) listens and responds to GCS events, for instance, file creation, modification, and deletion. CF, which can process images, change the format of videos, and run any services across the Internet; Webhooks, which respond to events from third-party systems such as Slack, Stripe, and GitHub, as well as any system capable of sending HTTP requests, using a simple HTTP trigger; Lightweight APIs make applications more lightweight with loosely coupled bits of logic that are simple to develop and scale rapidly. The functions could be either directly triggered through HTTP/S or event-driven; Users can write the mobile backend and mobile platform by using CF. Realtime database, storage, authentication, and Firebase Analytics events are all monitored and handled; Relating to the IoT area, as data is sent into Pub/Sub by tens of thousands or even hundreds of thousands of devices, cloud functions are launched, processing, transforming, and storing the data. With the help of Cloud Functions, users can do this entirely serverless. (Figure 4)

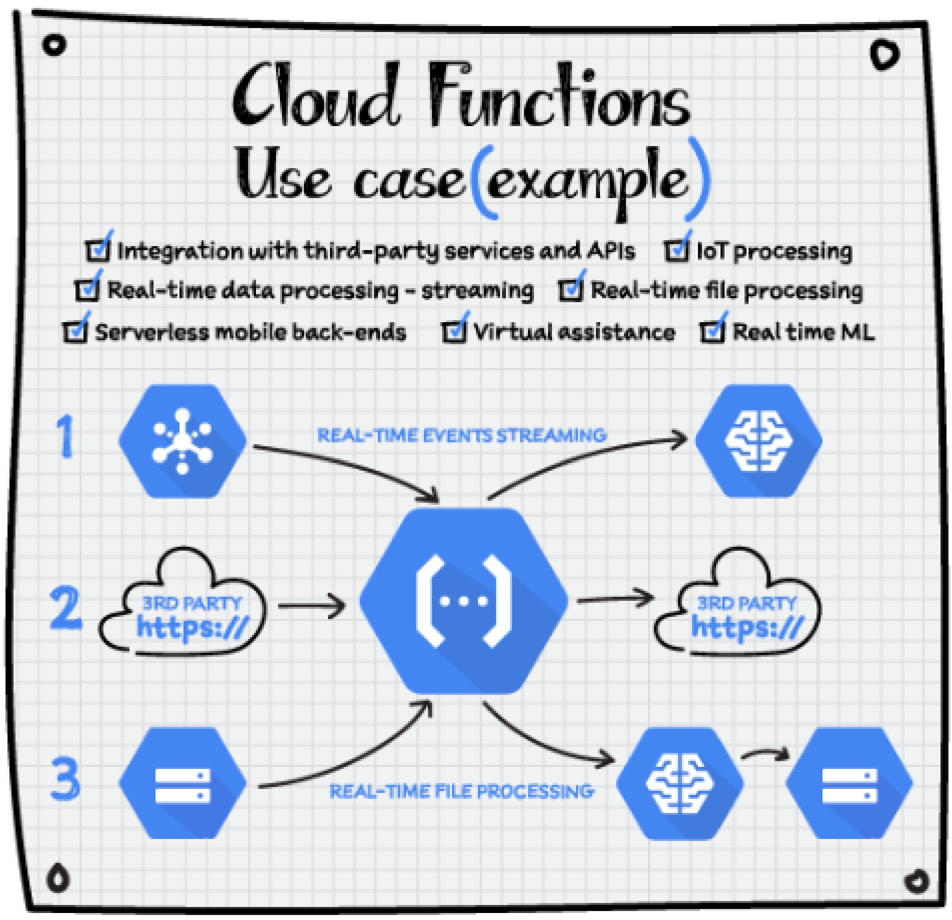

Cloud Functions
Use case (example)
Integration with third-party services and APIs
IoT processing
Real-time data processing - streaming
Real-time file processing
Serverless mobile back-ends
Virtual assistance
Real time ML
1
REAL-TIME EVENTS STREAMING
2
3RD PARTY https://
3RD PARTY https://
3
REAL-TIME FILE PROCESSING

Figure 4: Cloud Functions use cases [13]

2.4 Pub/Sub (P/S)

Cloud P/S [14] is an asynchronous messaging system that is both managed and scalable. Publisher, subject matter, and subscription make up its three constituent pieces. A message is sent by the publisher, and topics determine which messages are sent. A publisher and a subscriber can interact via messages sent to and received from the topic. The actual message is made up of separate sets of data and keys. Cloud P/S supports both push and pull subscription models, where messages can be sent to a subscriber or requested by the subscriber. All sent messages are stored until they receive a delivery confirmation from the recipient and are resent if necessary. Any message posted to a topic will be sent to all of its subscribers, and vice versa. Figure 5 is an example of how data flows through a Cloud P/S system.

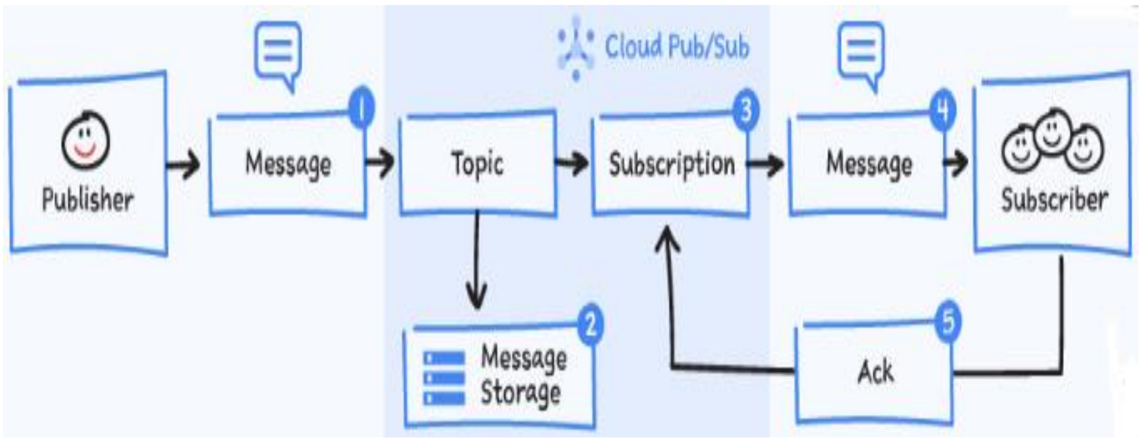


Figure 5: Cloud Pub/Sub message flow [15]

Cloud P/S can serve a variety of purposes in GCP, such as data streaming to BQ and asynchronous message in which a publisher is notified when a new file is uploaded to GCS. A solution for an easy-to-use asynchronous messaging system in GCP that has been extensively tested and is in use on a vast scale within Google. Cloud P/S is not an open-source technology. The use case limitation is that it is not intended for huge items due to the message size limit.

There are some use cases of Cloud P/S. For utilizing server events from the system or user interaction events from the end-user apps, users can transmit these events to P/S. Then, the events can be delivered to databases using a stream processing technology. BQ and Cloud Storage are examples of such databases. Cloud P/S enables simultaneous collecting of events from several clients.

Unprocessed or processed events can be made available for real-time processing to a variety of apps within teams and companies. Cloud P/S provides event-driven application design patterns and an enterprise event bus. Cloud P/S enables integration with numerous Google services which export the events to Cloud P/S. Data among databases are replicated. Cloud P/S is frequently utilised to deliver database modification event notifications. In BQ and other data storage systems, users can use these events to generate the database's state history and state representation.

Using Cloud P/S messages to make a connection with CF enables the efficient distribution of several jobs to multiple workers. Sending email alerts, testing models of AI, compressing text files, as well as reformatting images are examples of these tasks.

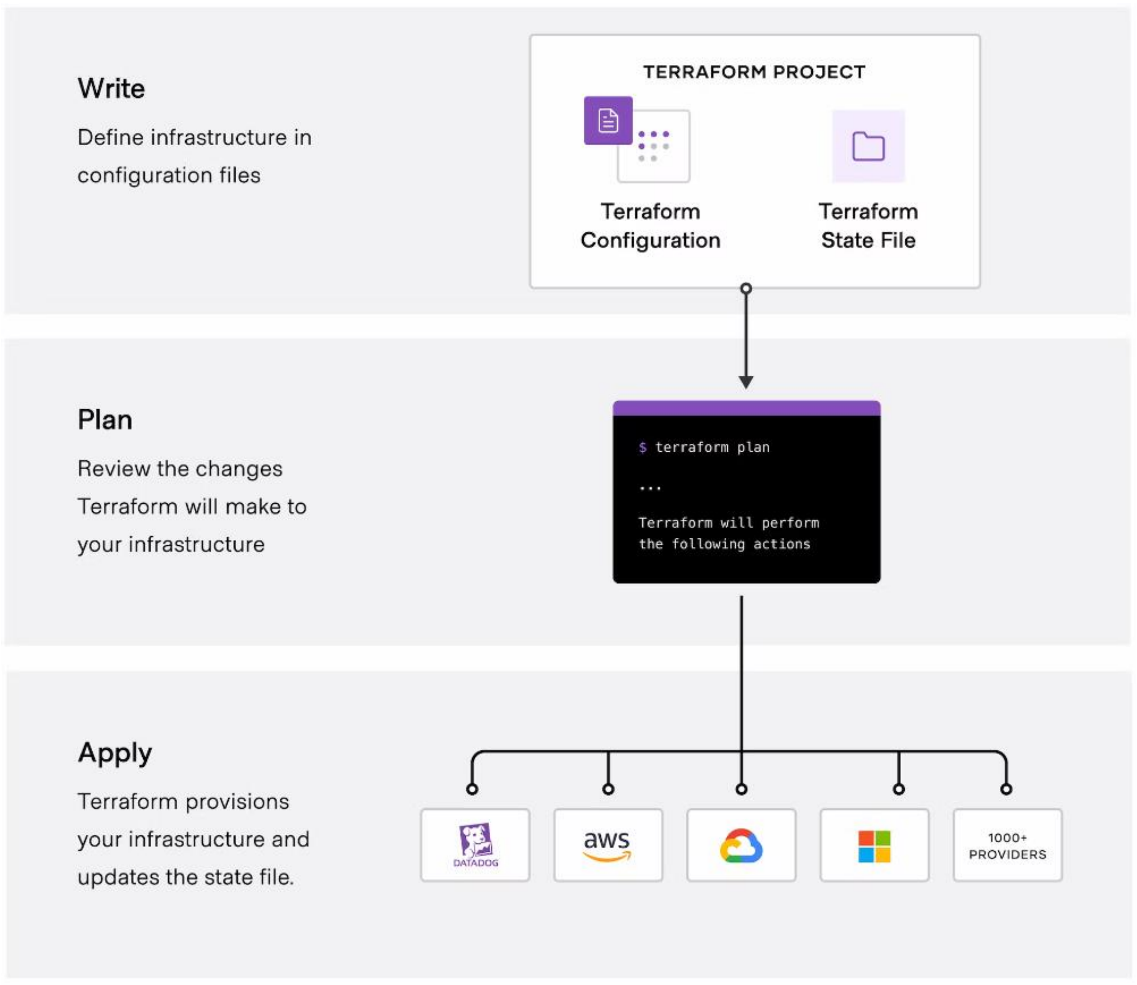
It is possible to build a real-time data sharing bus for the entire company, which can then be used to notify employees of database upgrades, analytics-related events, as well as business events.

Users can also implement Cloud P/S into data streaming from IoT devices and services. A real-time stream of events is one type of content that a SaaS application can deliver. The sensor devices in home can transmit data to Cloud P/S for use with other GCP products through dataflow pipeline. Refreshing distributed caches. Invalidation events can be published by an application to reflect changes to object identifiers. Load balancing for reliability. It's possible, for instance, for a service to have numerous Compute Engine deployments across different availability zones while still sharing a single topic of communication. If there is a problem with service in one region, the other regions can take over without any human intervention. [16]

2.5 Terraform

HashiCorp Terraform [17] is an infrastructure-as-code platform that enables the definition of both cloud and on-premises resources in human-readable configuration files that can be versioned, reused, and shared. Then, users can utilize a uniform procedure to provide and manage the whole infrastructure throughout its lifespan. Terraform can handle both low-level and high-level components, including computing, storage, and networking resources, DNS records, and SaaS capabilities.

Terraform generates and maintains cloud platform resources using their application programming interfaces (APIs). Providers allow Terraform to interact with practically any platform or service that has an API. (Figure 6)


Write
Define infrastructure in configuration files
TERRAFORM PROJECT
Terraform Configuration
Terraform State File
Plan
Review the changes Terraform will make to your infrastructure
Apply
Terraform provisions your infrastructure and updates the state file.
DATADOG
aws
1000+ PROVIDERS

```
$ terraform plan
...
Terraform will perform
the following actions
```

Figure 6: The stages of Terraform workflow

2.6 SendGrid API

SendGrid [18] is the world's biggest cloud-based email platform for delivering important messages. SendGrid's technology improves email deliverability, gives actionable insights, and expands to any email volume, freeing enterprises of the expense and complexity of managing specialized email infrastructures. SendGrid is the preferred email distribution platform for more than 150,000 online application firms and developers, such as Uber, Airbnb, Yelp, HubSpot, CBS

Interactive, Spotify, and Pandora. SendGrid sends over 200 billion emails annually.

2.7 Python

Python is a popular programming language in both conventional IT operations and DevOps owing to its mix of flexibility, power, and usability. Early in the 1990s, the Python programming language was made available for system management. It has been an enormous success in this field and has garnered widespread acceptance. Python is a general-purpose programming language that is used in almost every industry. It was welcomed by the visual effects and motion picture industries. Recent years have seen Python become the language of choice for data science and machine learning. It has been used in several fields, including aviation and bioinformatics. Python has a vast array of tools to accommodate the diverse demands of its users. [19]

2.8 Structured Query Language (SQL)

SQL is a programming language that is used to manage and process data in a relational database or in a relational data stream management system. It works especially well with structured data, like data that shows relationships between entities and variables. SQL is also the main way that users use to interact with BQ.

Query statements, referred to as Data Query Language statements, are the standard method for analysing data in BQ. After scanning tables or expressions, they will return rows of computed results. With Data Definition Language commands, users can make changes to database objects including rows, columns, tables, and views. Otherwise, to insert, update or delete data from BQ tables, users can utilise Data Manipulation Language statements. When users want to control the access and capacity of BQ system resources, they can use Data Control Language. The transactions can be managed by Transaction Control Language.

BQ supports SQL:2011 in its entirety, including arrays and sophisticated joins. BQ's ability to handle arrays eliminates the requirement to flatten nested and repeated fields when storing hierarchical data as JSON records. Not only does BQ support SQL:2011, but it also offers a few useful additions that expand its utility beyond the typical data warehouse contexts. Among these additions is the capacity to link tables based on distance or overlap criteria, as well as support for a variety of spatial functions that enable location-aware queries. As a result, BQ is enable to conduct descriptive analytics as power engine.

There are three main reasons why SQL was selected. First, anyone, not only programmers, can use SQL to address data issues. Additionally, SQL allows for any computation to be performed on the data. SQL is more than just a friendly and approachable language. Additionally, its power is substantial. Lastly, SQL always terminates. This means that datacenters can safely host SQL processing without fear of being taken over by an infinite loop.

2.9 Service Account (SA)

Applications or compute workload, such as a BQ, Cloud Functions and GCS for instance, use a special kind of account, which is called service account, to make authorized API calls. With domain-wide delegation, other Cloud Identity or Google Workspace users can make API calls while authenticated as the SA.

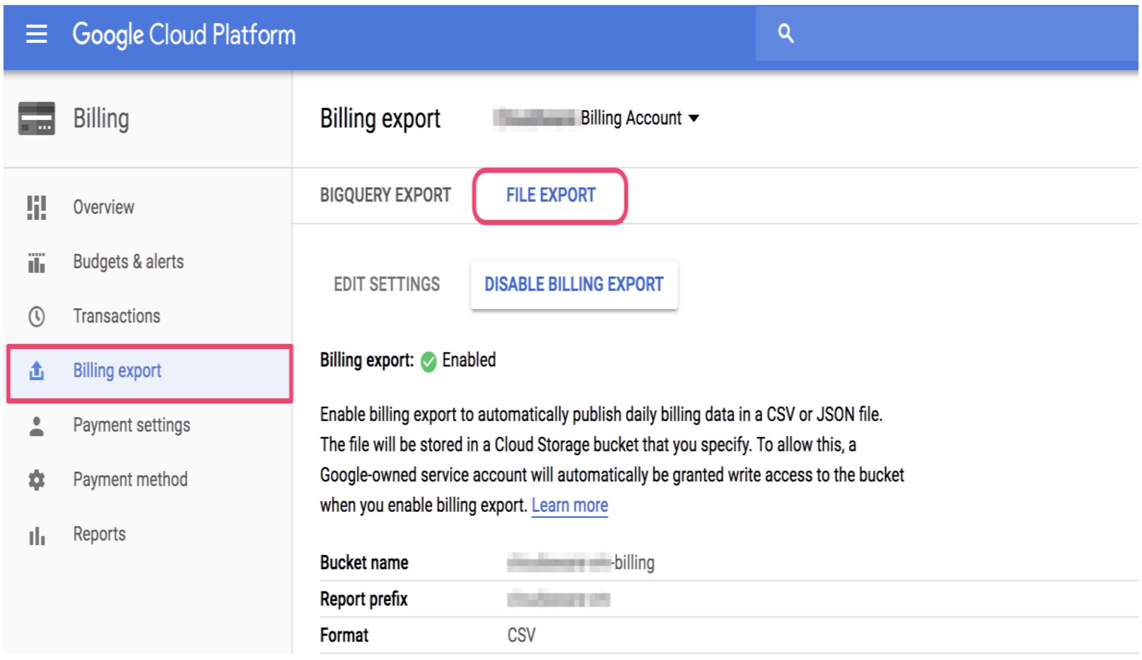
Particularly, if the applications running on a compute engine want to authenticate as a SA, it first needs to be attached to that BQ. The applications use their service account also in order to identify themselves. Furthermore, the roles of the SA manage which resources the apps can access by granting a SA IAM's roles that enable it to access the resources. The email address associated with a SA is its primary identifier.

Speaking of how a SA would differ from a user account, there are four main points that can be stated. Passwords do not exist in the use of SA, and logging in a SA via cookies or browsers is not possible. Signing data and authenticating with Google both require the usage of public and private RSA (Rivest-Shamir-Adleman) key pairs, which are associated with SA. It is possible for another user or SA to mimic a SA. Since a user can use a user account to share assets of Google Workspace (GW) like events or documents with their whole GW domain, this operation is restricted to SA. This is explained by SA not belonging to their GW domain, and the GW asset creation by SA not even happening within the GW domain. Therefore, any of these assets cannot be controlled by GW and Cloud Identity's admins. Still, one important thing needs to be noticed that those assets which are created in GW domain, they are created when using domain-wide delegation. Based on the above-provided information, with the goal of enabling such a possibility, it is possible to understand that API requests are approved as those of the impersonated user rather than the SA itself.

3 Solution Design

3.1 Google Billing Export Solutions

Before 2017, GCP only offered the billing export to CSV or JSON files method for the Google Cloud's users. This method allowed users to export the billing data which including the price, usage as well as the cost estimates directly to Cloud Storage Bucket. It was really simple to use. Users needed to fulfil the bucket name, report prefix and files format like CSV or JSON. (Figure 7)



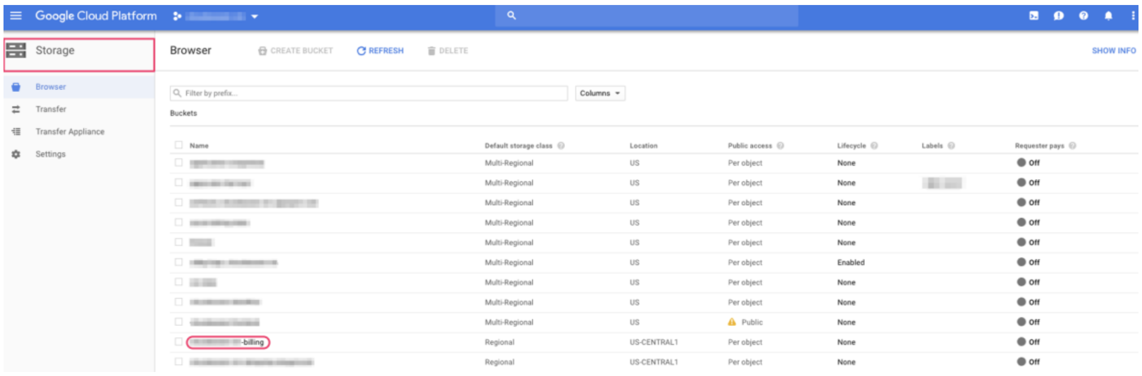
Billing export: Enabled

Enable billing export to automatically publish daily billing data in a CSV or JSON file. The file will be stored in a Cloud Storage bucket that you specify. To allow this, a Google-owned service account will automatically be granted write access to the bucket when you enable billing export. Learn more

Bucket name	[redacted]-billing
Report prefix	[redacted]
Format	CSV

Figure 7: Billing export via file export

All exported files would be available on Cloud Storage Bucket. Within 24 hours of reporting's activation, billing reports will be available in the 'Objects' section. (Figure 8)

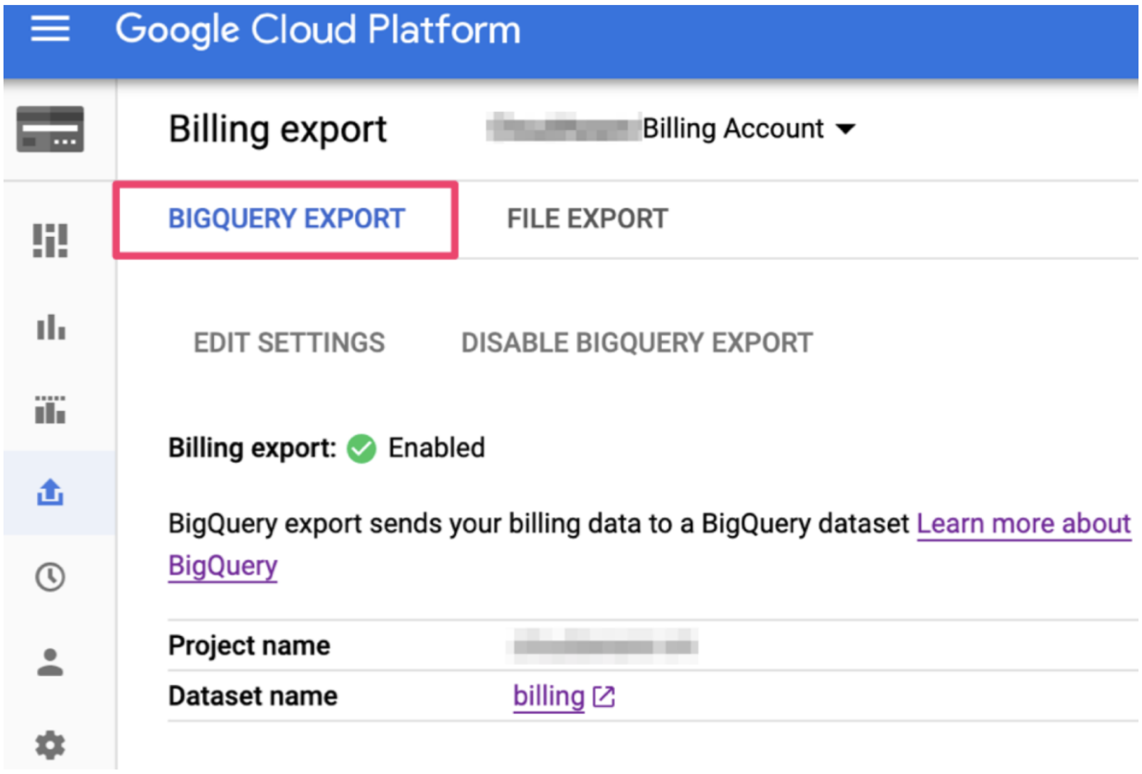


Name	Default storage class	Location	Public access	Lifecycle	Labels	Requester pays
[redacted]	Multi-Regional	US	Per object	None		Off
[redacted]	Multi-Regional	US	Per object	None	[redacted]	Off
[redacted]	Multi-Regional	US	Per object	None		Off
[redacted]	Multi-Regional	US	Per object	None		Off
[redacted]	Multi-Regional	US	Per object	None		Off
[redacted]	Multi-Regional	US	Per object	Enabled		Off
[redacted]	Multi-Regional	US	Per object	None		Off
[redacted]	Multi-Regional	US	Per object	None		Off
[redacted]	Multi-Regional	US	Public	None		Off
[redacted]-billing	Regional	US-CENTRAL1	Per object	None		Off
[redacted]	Regional	US-CENTRAL1	Per object	None		Off

Figure 8: Billing data was available on GCS as CSV file.

In the end of 2017, GCP was pleased to announce the wide availability of billing export to BQ, the data warehouse service, providing customers with a more detailed and immediate view of GCP charges than ever before. The billing export to BQ is a new and enhanced version of the billing export to CSV or JSON files, and as the name implies, it exports cloud consumption data straight into a BQ dataset.

Additionally, the BQ billing export includes a few additional data-organizational features such as Utilize identifiers to classify and monitor expenses; Extra product information should be organized by GCP services; Service description; Class of service; SKU ID to identify each resource type in a unique manner; Export time to assist in cost invoice organization. (Figure 9)



Google Cloud Platform

Billing export [redacted] Billing Account

BIGQUERY EXPORT FILE EXPORT

EDIT SETTINGS DISABLE BIGQUERY EXPORT

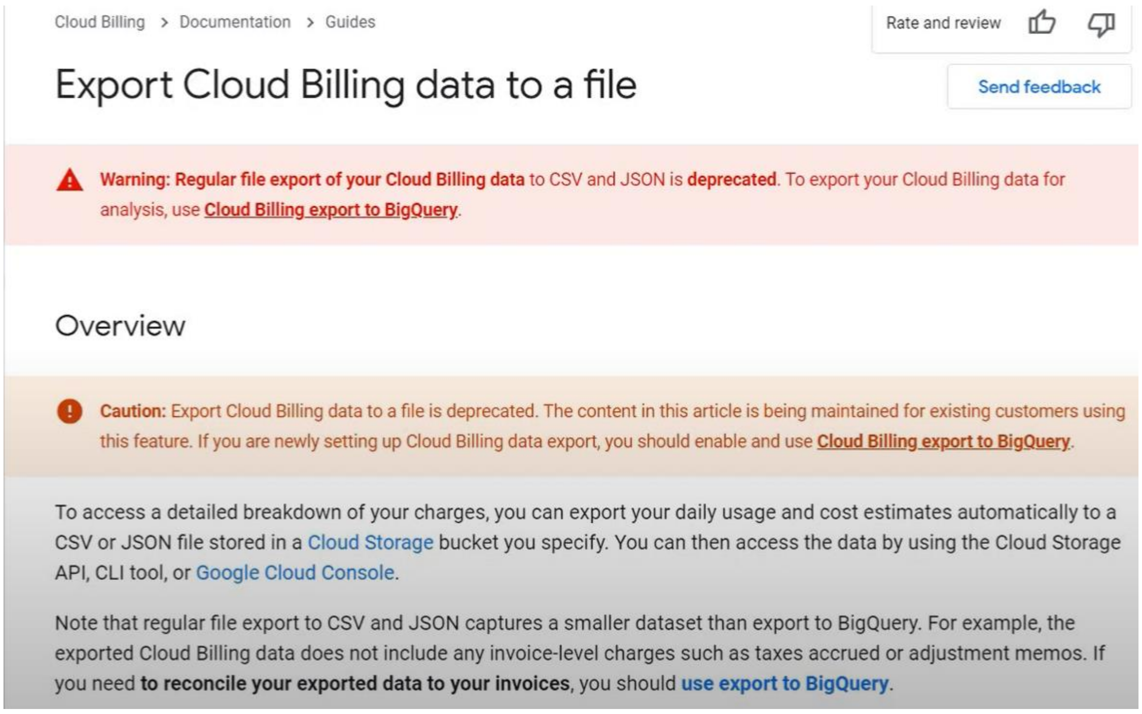
Billing export: ✓ Enabled

BigQuery export sends your billing data to a BigQuery dataset Learn more about BigQuery

Project name	[redacted]
Dataset name	billing

Figure 9: Billing export via BQ export

However, GCP still supported for the file export solution. This feature was deprecated in the beginning of 2021. GCP recommends exporting billing data to BQ because it contains much more information than file export. (Figure 10)


Cloud Billing > Documentation > Guides

Rate and review

Send feedback

Export Cloud Billing data to a file

Warning: Regular file export of your Cloud Billing data to CSV and JSON is **deprecated**. To export your Cloud Billing data for analysis, use Cloud Billing export to BigQuery.

Overview

Caution: Export Cloud Billing data to a file is deprecated. The content in this article is being maintained for existing customers using this feature. If you are newly setting up Cloud Billing data export, you should enable and use Cloud Billing export to BigQuery.

To access a detailed breakdown of your charges, you can export your daily usage and cost estimates automatically to a CSV or JSON file stored in a Cloud Storage bucket you specify. You can then access the data by using the Cloud Storage API, CLI tool, or Google Cloud Console.

Note that regular file export to CSV and JSON captures a smaller dataset than export to BigQuery. For example, the exported Cloud Billing data does not include any invoice-level charges such as taxes accrued or adjustment memos. If you need **to reconcile your exported data to your invoices**, you should **use export to BigQuery**.

Figure 10: Export Cloud Billing data to a file has been deprecated by Google.

3.2 Recommendation Billing Export Solution

The purpose of this internal project was leveraging the benefits of billing data export to BQ function and the GCP services to build the automated export. At first, the architecture diagram was drawn based on the GCP components after defining their functions. This solution was built step-by-step manually before implementing automatically.

The architecture for this project is shown in Figure 11 which including functional steps:

- BQ was configured with a scheduled query.
- Every successful scheduled query execution triggered a Pub/Sub topic.

- A Cloud Function subscribed to a Pub/Sub topic and exports query results to GCS with a job ID prefix of email export. The GCS bucket would always contain the most recent export, and this file would be replaced with each subsequent export.
- When a new object was added to a specified bucket, GCS triggered a second topic through Pub/Sub.
- A second function in the Cloud Function subscribed to the Pub/Sub topic described in the previous section and sent the email using the SendGrid API with a link to the signed or unsigned URL of the file.
- The SendGrid API was a web-based API that sends users an email including a signed URL.

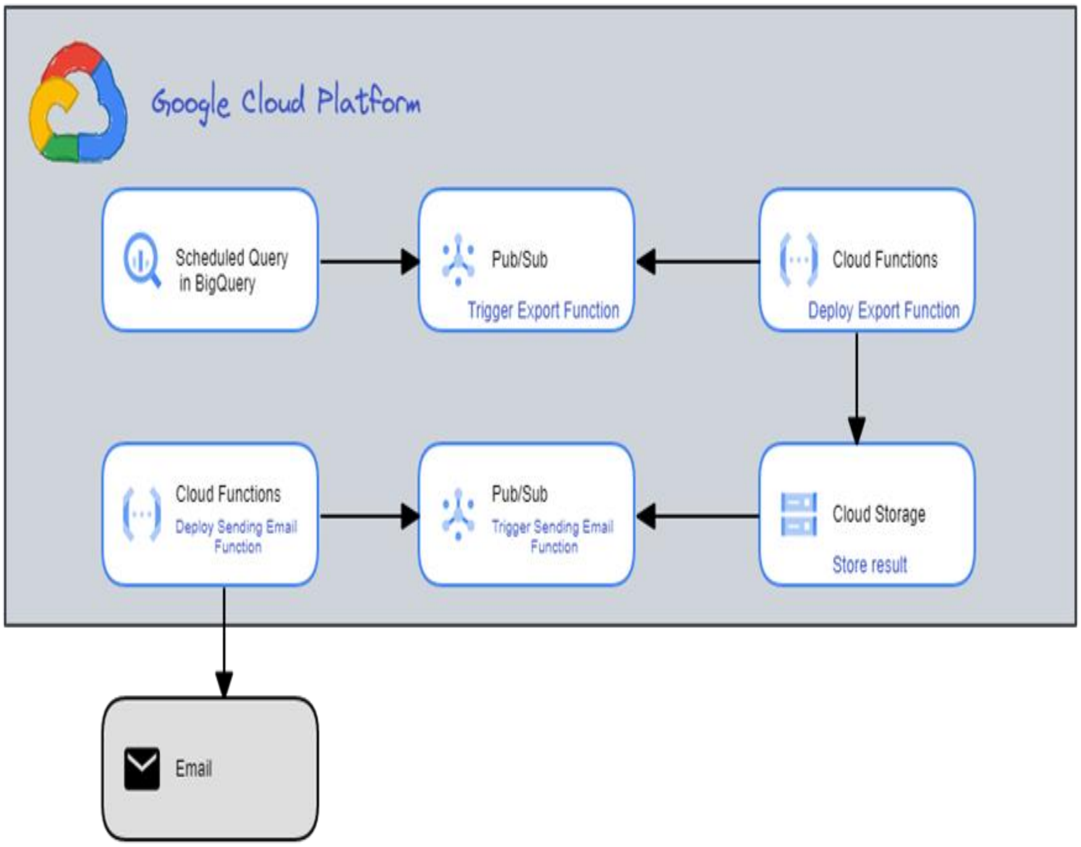
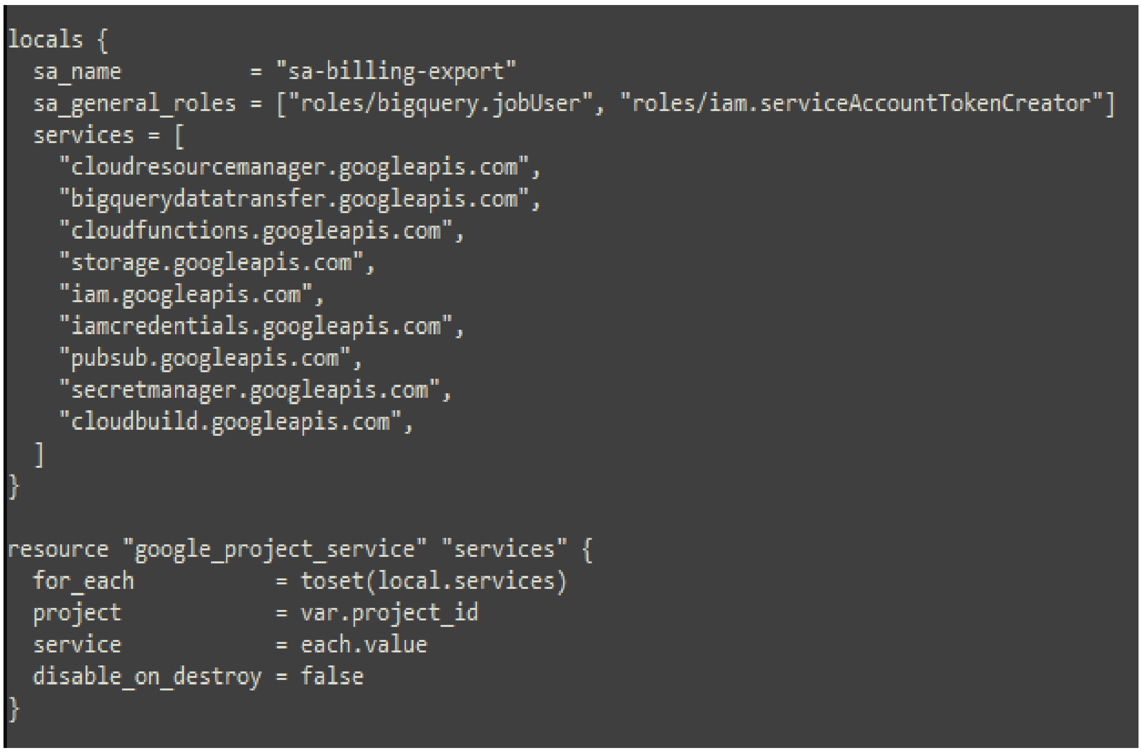


Figure 11: Solution Architecture Diagram.

4 Implementation

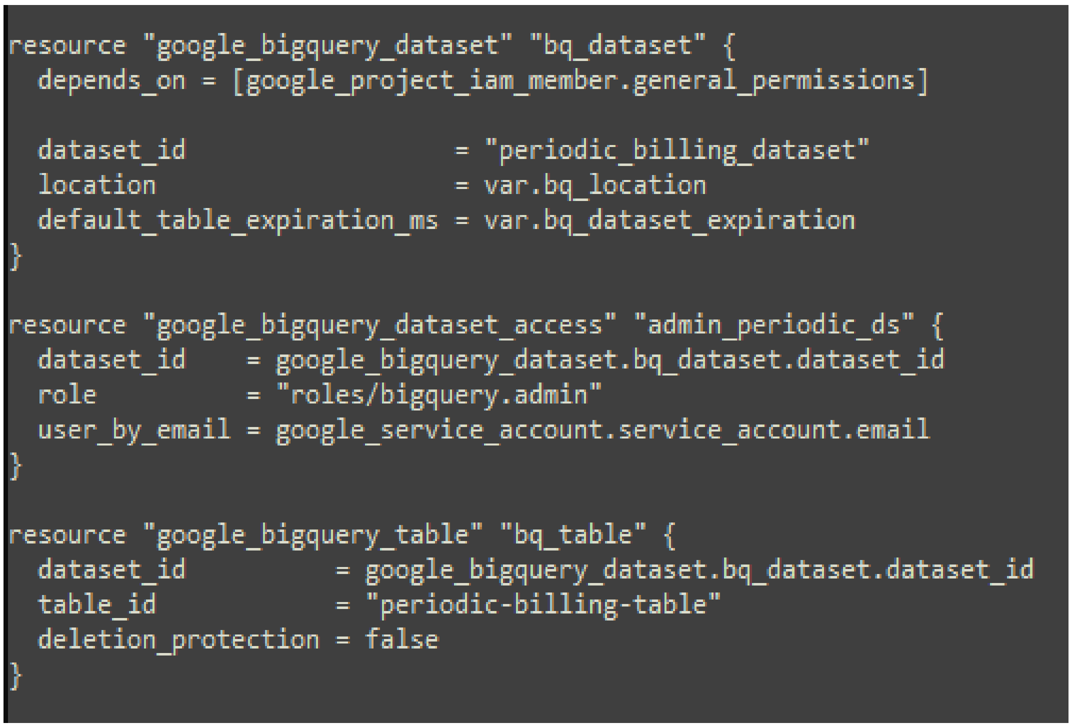
4.1 Setup

Frist, all relating google service APIs (Application programming interface) need to be checked and enabled for services to be available. It is shown in the Terraform code bellows. In this project, the google services API include Cloud Resource Manager API, BigQuery Data Transfer API, Cloud Functions API, Cloud Storage API, Identity and Access Management (IAM) API, IAM Service Account Credentials API, Pub/Sub API, Secret Manager API and Cloud Build API. These APIs can be found in Google API Library. Each API run on subdomains googleapi.com over public Internet and virtual networks.



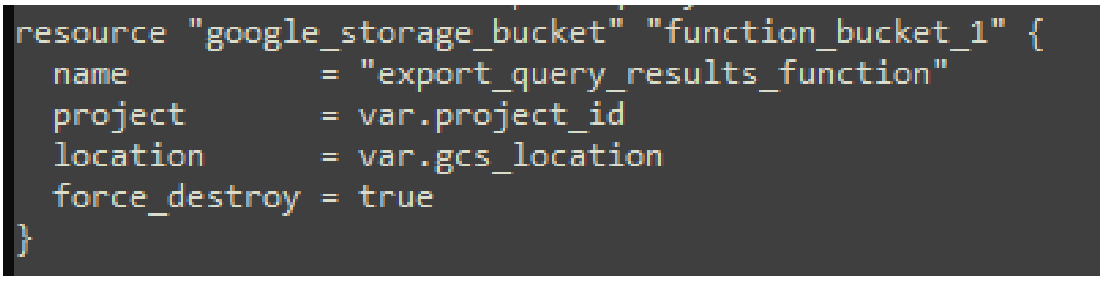
```
locals {
  sa_name          = "sa-billing-export"
  sa_general_roles = ["roles/bigquery.jobUser", "roles/iam.serviceAccountTokenCreator"]
  services = [
    "cloudresourcemanager.googleapis.com",
    "bigquerydatatransfer.googleapis.com",
    "cloudfunctions.googleapis.com",
    "storage.googleapis.com",
    "iam.googleapis.com",
    "iamcredentials.googleapis.com",
    "pubsub.googleapis.com",
    "secretmanager.googleapis.com",
    "cloudbuild.googleapis.com",
  ]
}

resource "google_project_service" "services" {
  for_each           = toset(local.services)
  project            = var.project_id
  service            = each.value
  disable_on_destroy = false
}
```

Create a BQ dataset to store the tables produced for each export. If customers want to get an email every day, for instance, this dataset would contain a table for each daily export with a naming convention such as "daily_export_$TIMESTAMP." The suggestion is to select a default table expiry period for this dataset due to its potential for rapid growth. This enables the removal of tables containing outdated data. The minimum lifetime of all dataset tables is 3600000 milliseconds. Changes to this property's value will only influence the creation of new tables and not impact existing ones. Whenever a table's expiration time is reached, the table in question will be removed without human intervention. A table's default expiration time is indicated by this property, but any specific expirationTime provided when creating a table, or any time the expirationTime property is changed or deleted, even before the table expires, will be used instead. Relating to dataset locations, BQ supports region and multi-region. Multi-regions contain two or more places such as EU and US. Region is only one specific area, for example, Iowa, Oregon in Americas, Melbourne, Delhi in Asia Pacific, and Belgium, Finland in Europe. Once the dataset was created, the location cannot be modified. Normally, the US multi-region will be set as the default. If the requested datasets are not in the same location as those provided, BQ will throw an error. Every read and written dataset included in the request must have the same location as the job, whether that location is inferred or defined. The deletion_protection argument prevents Terraform destroy the instance.



```
resource "google_bigquery_dataset" "bq_dataset" {
  depends_on = [google_project_iam_member.general_permissions]

  dataset_id                  = "periodic_billing_dataset"
  location                    = var.bq_location
  default_table_expiration_ms = var.bq_dataset_expiration
}

resource "google_bigquery_dataset_access" "admin_periodic_ds" {
  dataset_id    = google_bigquery_dataset.bq_dataset.dataset_id
  role          = "roles/bigquery.admin"
  user_by_email = google_service_account.service_account.email
}

resource "google_bigquery_table" "bq_table" {
  dataset_id          = google_bigquery_dataset.bq_dataset.dataset_id
  table_id            = "periodic-billing-table"
  deletion_protection = false
}
```

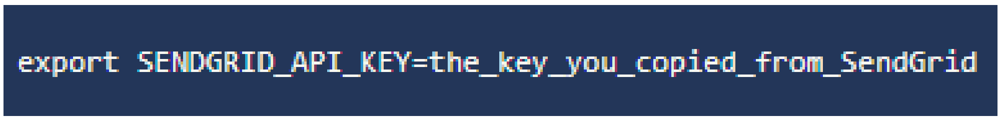
Next, create a Cloud Storage bucket to store the CSV files exported by BQ. Based on the "Age" condition, the bucket lifecycle management settings can automatically delete CSV files or move them to a new storage class. This is similar to how the dataset expiration time works. The location must be set same as the location of BQ dataset. If the BQ dataset is in Finland (europe-north1), the GCS bucket must be in Finland region. The force_destroy is true that mean when a bucket is removed, all of the objects which are in it will also be deleted.



```
resource "google_storage_bucket" "function_bucket_1" {
  name          = "export_query_results_function"
  project       = var.project_id
  location      = var.gcs_location
  force_destroy = true
}
```

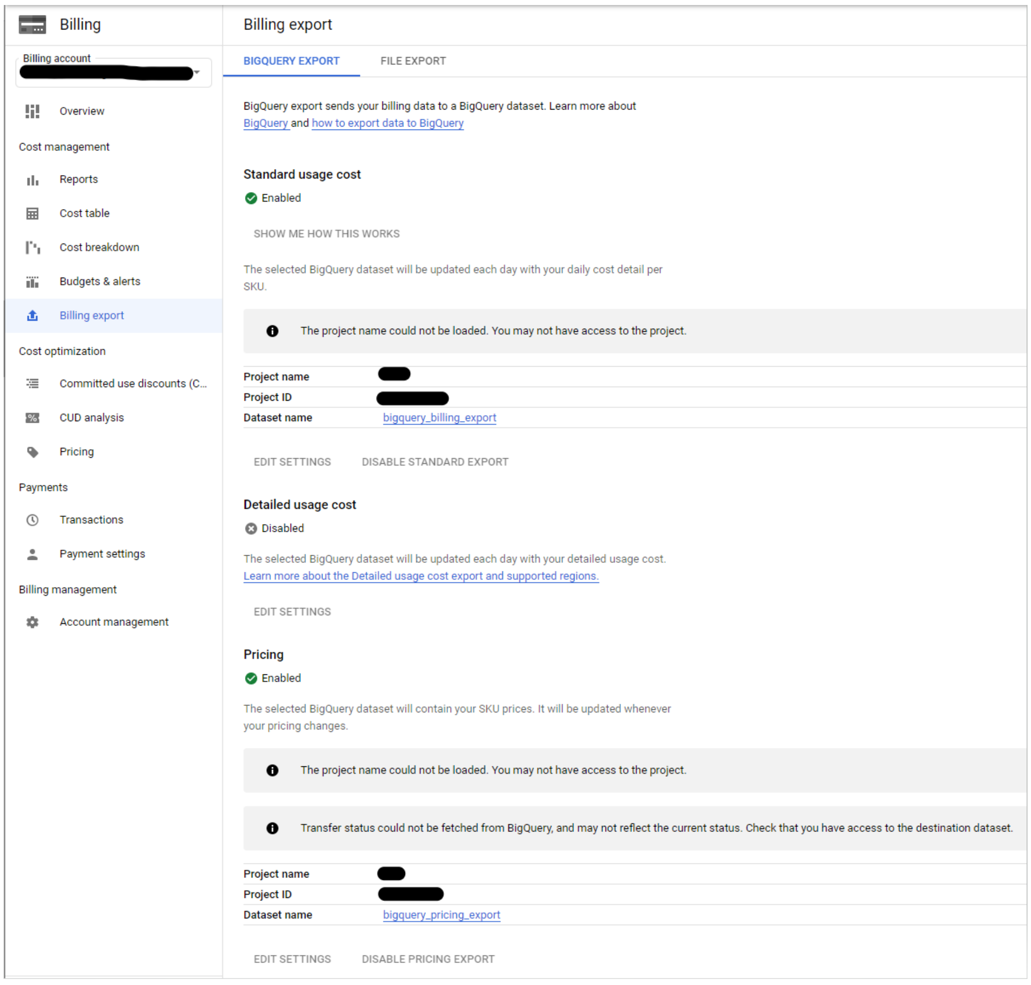
The next configuration step includes enabling access to the SendGrid API. Create an account and a SendGrid API key to enable the Cloud Function to authenticate with the Email API and send an email. SendGrid's free plan lets users send

40,000 messages a day for the first 30 days. After that, users can send 100 messages a day forever. After API key was created, it should be stored in the environment variable or config file. It makes sure that the value can be viewed by those who have permission to access. As a result, they are a safer option for storing sensitive information like API keys.



```
export SENDGRID_API_KEY=the_key_you_copied_from_SendGrid
```

To export the billing data to the BQ dataset, the billing export function in Cloud Billing must be enabled. There are three billing data tables such as standard usage cost, pricing, and detailed cost. It depends on the requirements of customers for choosing one of these. After that, the data will be exported automatically to the BQ dataset that was created above. With the standard and detailed usage cost export, the loading process will take a few hours. If customers select the pricing export or standard usage cost, the billing data only includes the information for charges made on or after the enabled day. The billing will be available from the beginning of the prior month, if the BQ dataset location is multi-region. For instance, on July 23, detailed use cost data export will be enabled for data from June 1. If the BQ dataset is a region location, the precise consumption cost data is available after billing export. That is, the billing data is not updated retrospectively for non-multi-region locations dataset. The billing data can be unavailable for clients who have re-enabled, enabled, as well as disabled detailed use cost data export. The exported data cannot be recovered if it is removed.



BigQuery export sends your billing data to a BigQuery dataset. Learn more about BigQuery and how to export data to BigQuery

Standard usage cost
Enabled

The selected BigQuery dataset will be updated each day with your daily cost detail per SKU.

The project name could not be loaded. You may not have access to the project.

Project name	[redacted]
Project ID	[redacted]
Dataset name	bigquery_billing_export

Detailed usage cost
Disabled

The selected BigQuery dataset will be updated each day with your detailed usage cost. Learn more about the Detailed usage cost export and supported regions.

Pricing
Enabled

The selected BigQuery dataset will contain your SKU prices. It will be updated whenever your pricing changes.

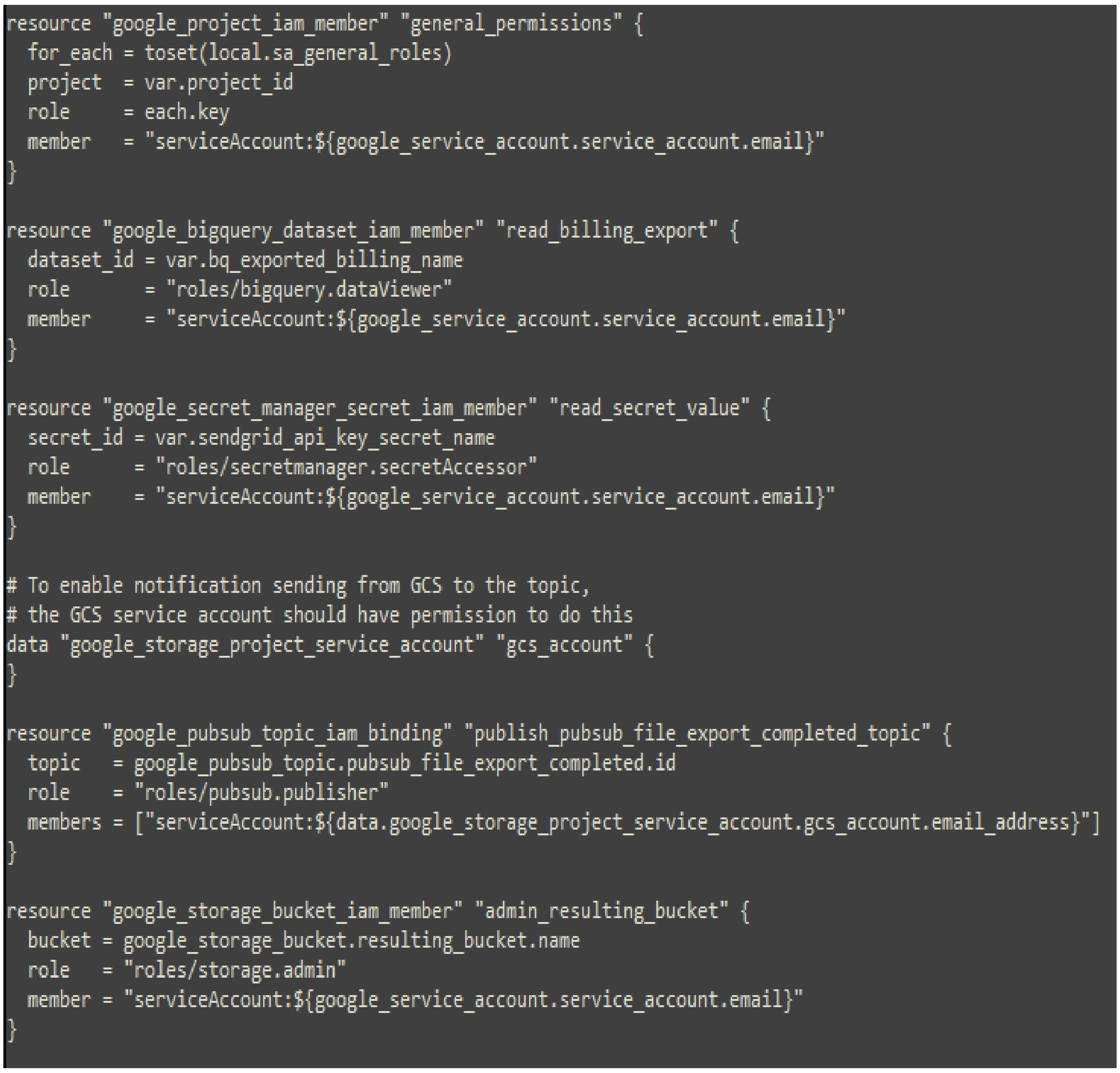
The project name could not be loaded. You may not have access to the project.

Transfer status could not be fetched from BigQuery, and may not reflect the current status. Check that you have access to the destination dataset.

Project name	[redacted]
Project ID	[redacted]
Dataset name	bigquery_pricing_export

4.2 Create service account

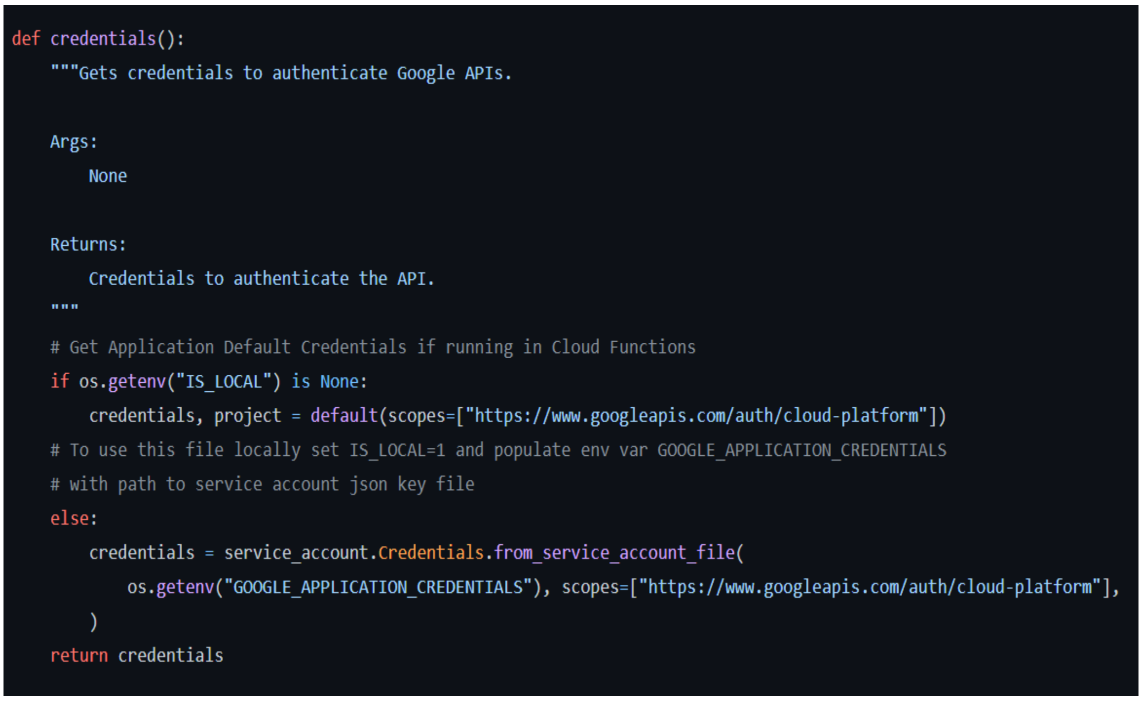
SendGrid API authentication requires the creation of a service account. For the service account to generate signed credentials for Cloud Functions, it requires the Service Account Token Creator role. The BQ Admin and Storage Object Admin roles must be added so that the user can perform BQ and Storage tasks. The google_project_iam_member, google_bigquery_dataset_iam_member, google_secret_manager_secret_iam_member and google_storage_bucket_iam_member resources are non authoritative. The role of a new member will be granted by updating the IAM policy via these resources such as BQ data viewer, Secret Manager secret assessor, Pub/Sub polisher and Storage admin. Otherwise, they will preserve the role of others. To enable notification sending from GCS to the topic, the GCS service account should have permission to do this. The following code sample establishes a service account with the aforementioned roles:



```
resource "google_project_iam_member" "general_permissions" {
  for_each = toset(local.sa_general_roles)
  project  = var.project_id
  role     = each.key
  member   = "serviceAccount:${google_service_account.service_account.email}"
}

resource "google_bigquery_dataset_iam_member" "read_billing_export" {
  dataset_id = var.bq_exported_billing_name
  role       = "roles/bigquery.dataViewer"
  member     = "serviceAccount:${google_service_account.service_account.email}"
}

resource "google_secret_manager_secret_iam_member" "read_secret_value" {
  secret_id = var.sendgrid_api_key_secret_name
  role      = "roles/secretmanager.secretAccessor"
  member    = "serviceAccount:${google_service_account.service_account.email}"
}

# To enable notification sending from GCS to the topic,
# the GCS service account should have permission to do this
data "google_storage_project_service_account" "gcs_account" {
}

resource "google_pubsub_topic_iam_binding" "publish_pubsub_file_export_completed_topic" {
  topic   = google_pubsub_topic.pubsub_file_export_completed.id
  role    = "roles/pubsub.publisher"
  members = ["serviceAccount:${data.google_storage_project_service_account.gcs_account.email_address}"]
}

resource "google_storage_bucket_iam_member" "admin_resulting_bucket" {
  bucket = google_storage_bucket.resulting_bucket.name
  role   = "roles/storage.admin"
  member = "serviceAccount:${google_service_account.service_account.email}"
}
```

4.3 The Cloud Functions codes

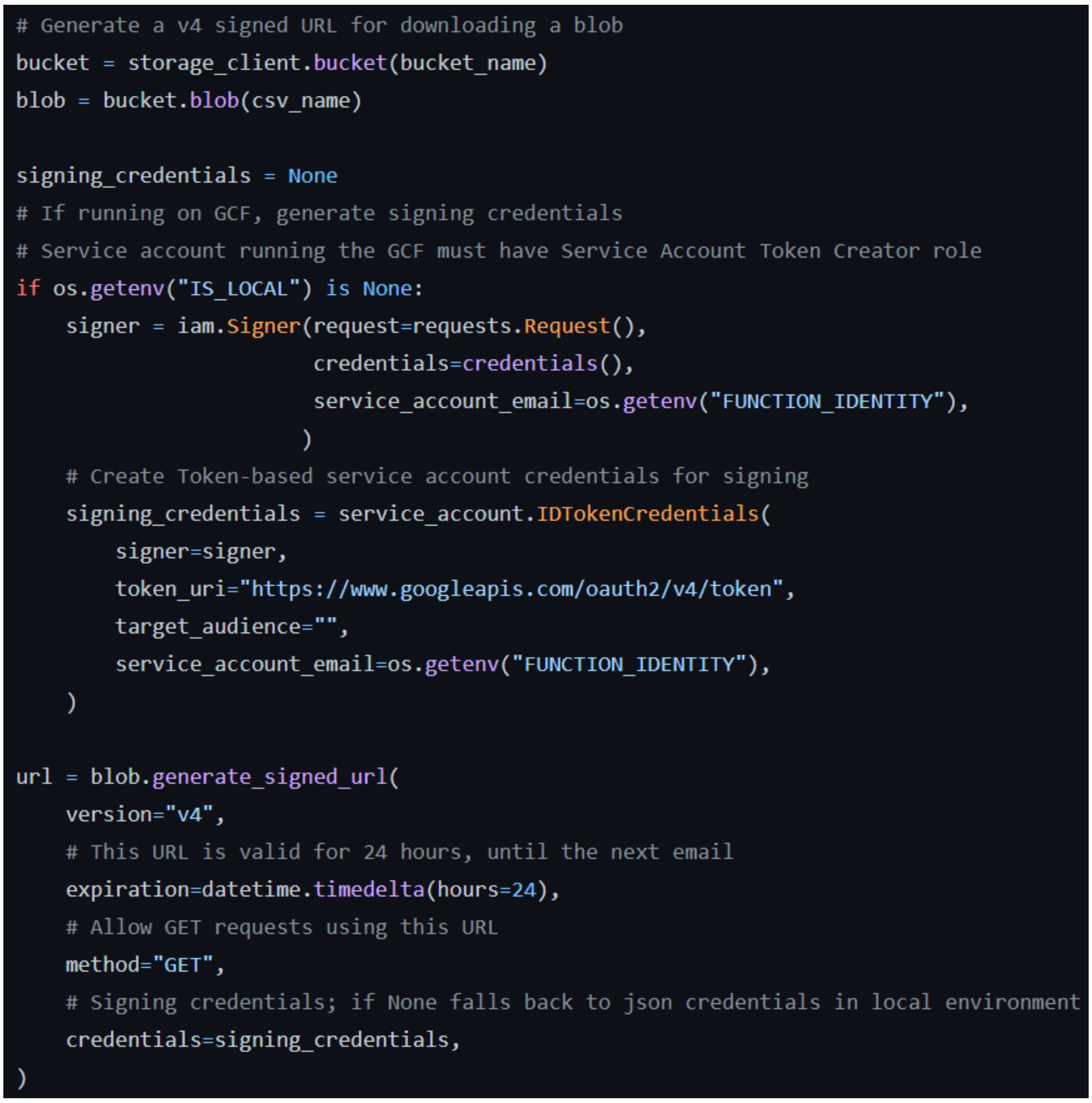
To implement this solution, invoke the Google BQ and Cloud Storage APIs using the Python Client Library. Instantiate the client library with the relevant service account credentials to ensure appropriate authentication and completion of the required tasks. If the primary script is to be executed in Cloud Functions, the credentials default to the Application Default Credentials. If the application is being executed locally, the GOOGLE_APPLICATION_CREDENTIALS as an environment variable will be used for reading the SA key file. The code sample below explains how to generate credentials:



```python
def credentials():
    """Gets credentials to authenticate Google APIs.

    Args:
        None

    Returns:
        Credentials to authenticate the API.
    """
    # Get Application Default Credentials if running in Cloud Functions
    if os.getenv("IS_LOCAL") is None:
        credentials, project = default(scopes=["https://www.googleapis.com/auth/cloud-platform"])
    # To use this file locally set IS_LOCAL=1 and populate env var GOOGLE_APPLICATION_CREDENTIALS
    # with path to service account json key file
    else:
        credentials = service_account.Credentials.from_service_account_file(
            os.getenv("GOOGLE_APPLICATION_CREDENTIALS"), scopes=["https://www.googleapis.com/auth/cloud-platform"],
        )
    return credentials
```

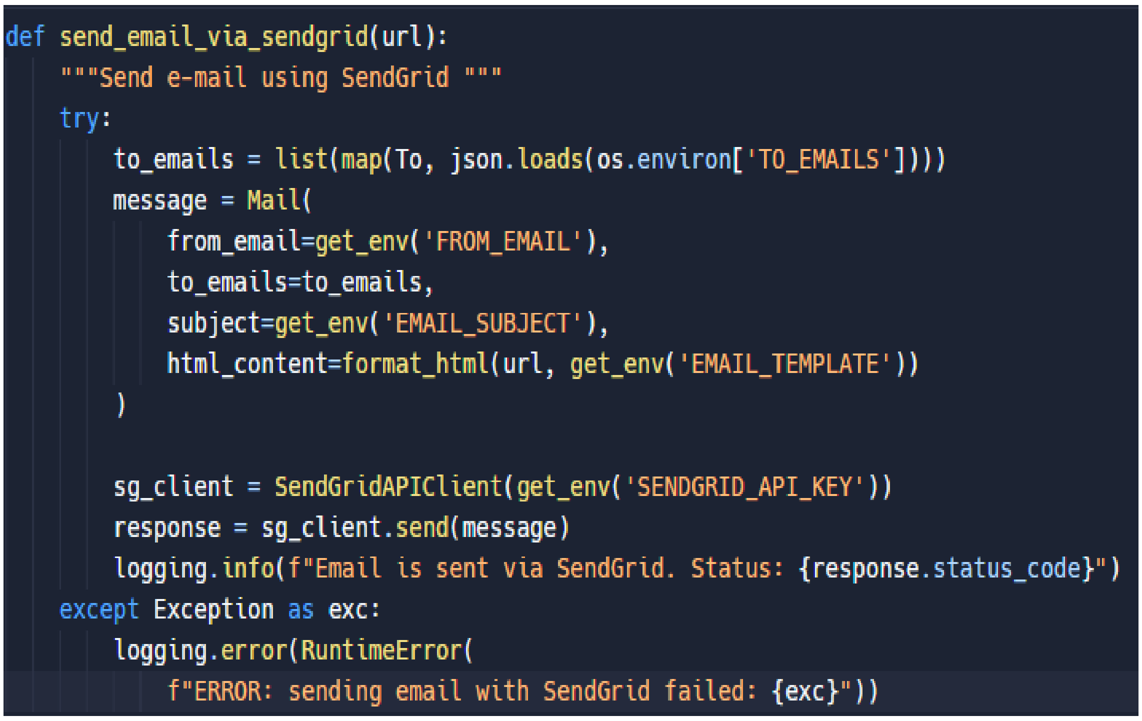
Customers can use the BQ and Cloud Storage client libraries to create a table, output the results of a query to it, and then export the table data as a CSV to Cloud Storage.

Create the signed URL for the CSV file that is stored in the bucket. This procedure comprises establishing an expiry time reflecting the link's accessibility duration. To prevent receivers from obtaining outdated material, the expiry time should be set to the time gap between emails. Function Identity retrieves the function's current identity for authentication purposes (the service account executing the function). iam.Signer() sends a request to the service account given to create OAuth credentials for the generate_signed_url() method.



```python
# Generate a v4 signed URL for downloading a blob
bucket = storage_client.bucket(bucket_name)
blob = bucket.blob(csv_name)

signing_credentials = None
# If running on GCF, generate signing credentials
# Service account running the GCF must have Service Account Token Creator role
if os.getenv("IS_LOCAL") is None:
    signer = iam.Signer(request=requests.Request(),
                        credentials=credentials(),
                        service_account_email=os.getenv("FUNCTION_IDENTITY"),
                       )
    # Create Token-based service account credentials for signing
    signing_credentials = service_account.IDTokenCredentials(
        signer=signer,
        token_uri="https://www.googleapis.com/oauth2/v4/token",
        target_audience="",
        service_account_email=os.getenv("FUNCTION_IDENTITY"),
    )

url = blob.generate_signed_url(
    version="v4",
    # This URL is valid for 24 hours, until the next email
    expiration=datetime.timedelta(hours=24),
    # Allow GET requests using this URL
    method="GET",
    # Signing credentials; if None falls back to json credentials in local environment
    credentials=signing_credentials,
)
```

To send the email using the SendGrid API, use the token that created and the SendGrid implementation instructions for the web API. The following example illustrates what this might look like:



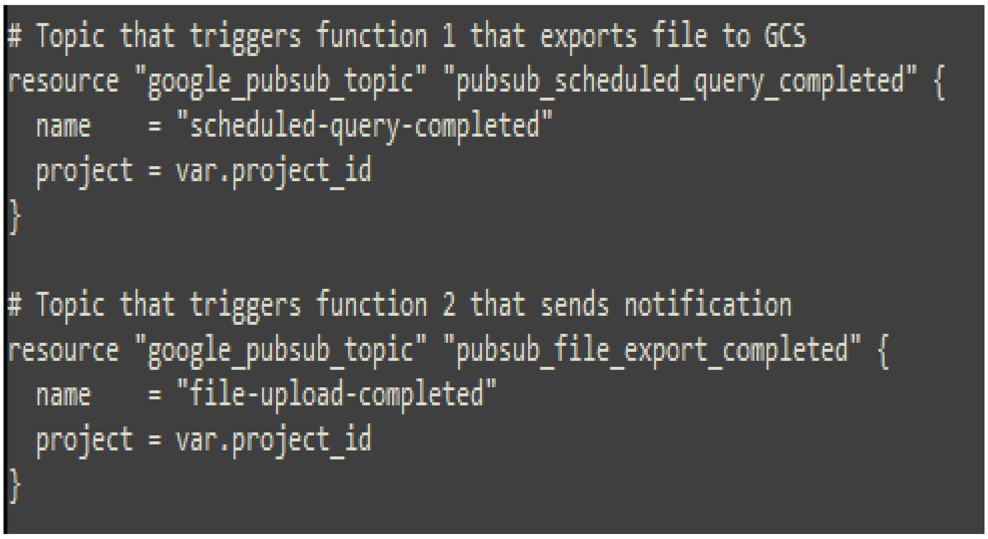
```python
def send_email_via_sendgrid(url):
    """Send e-mail using SendGrid """
    try:
        to_emails = list(map(To, json.loads(os.environ['TO_EMAILS'])))
        message = Mail(
            from_email=get_env('FROM_EMAIL'),
            to_emails=to_emails,
            subject=get_env('EMAIL_SUBJECT'),
            html_content=format_html(url, get_env('EMAIL_TEMPLATE'))
        )

        sg_client = SendGridAPIClient(get_env('SENDGRID_API_KEY'))
        response = sg_client.send(message)
        logging.info(f"Email is sent via SendGrid. Status: {response.status_code}")
    except Exception as exc:
        logging.error(RuntimeError(
            f"ERROR: sending email with SendGrid failed: {exc}"))
```

4.4 Deploy

Creating a Pub/Sub topic, deploying the Cloud Function with the code from the previous part, and setting up Cloud Scheduler to initiate the pipeline are the steps required to construct the pipeline via Terraform code.

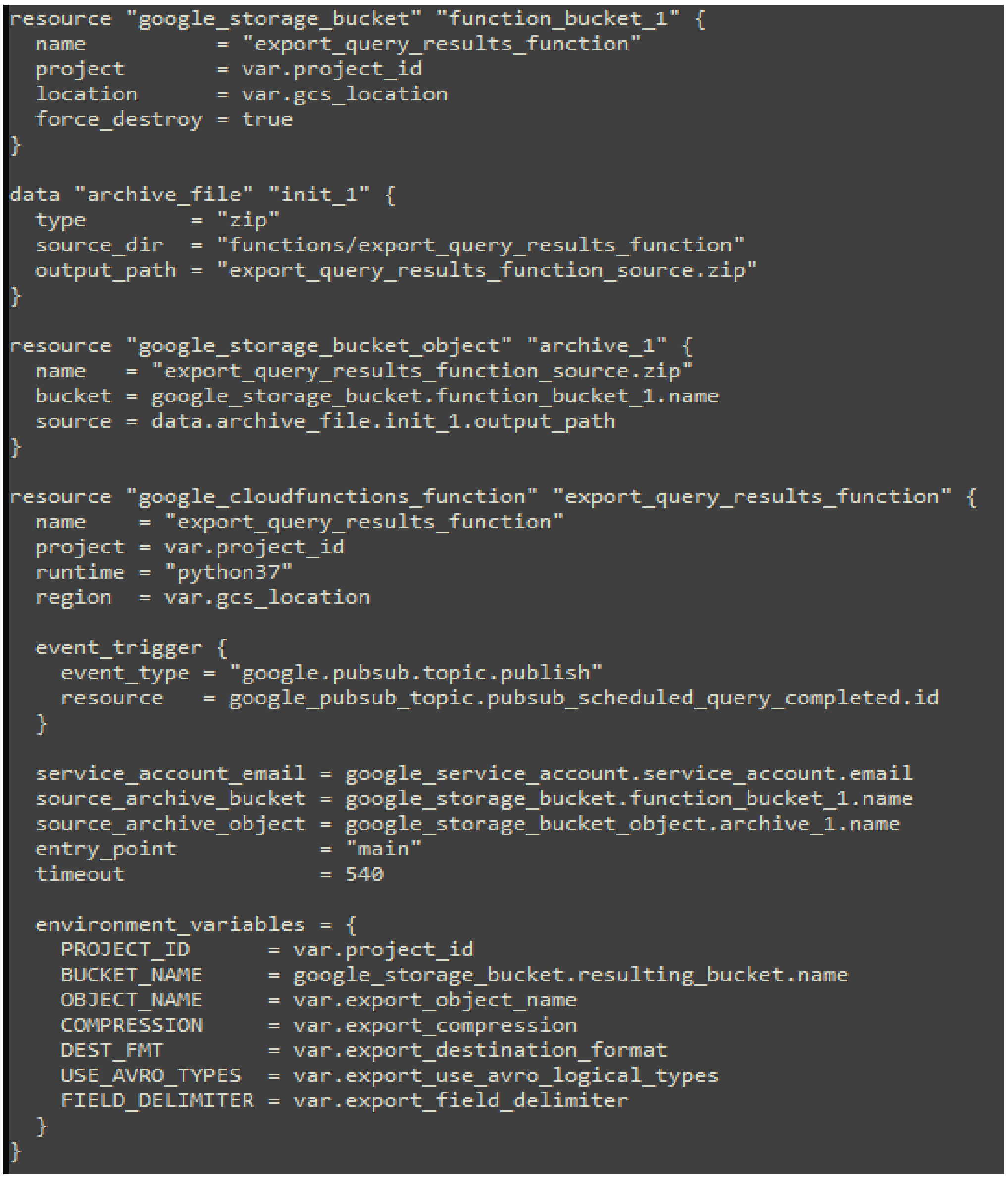
The Pub/Sub topics are crucial part in the pipeline. The first pub/sub topic namely pubsub_scheduled_query_completed triggers the function event of the Cloud Functions to notify the BQ when the billing data is imported. After that, the function runs on Cloud Functions to generate the schedule query results. These results will be compressed as achieve files and exported to Google Cloud

Storage. The second pub/sub topic call pubsub_file_export_completed will trigger the send email function to notify the GCS when the schedule query results are imported. Then, the second function will run on CF to send the email with signed url to clients.

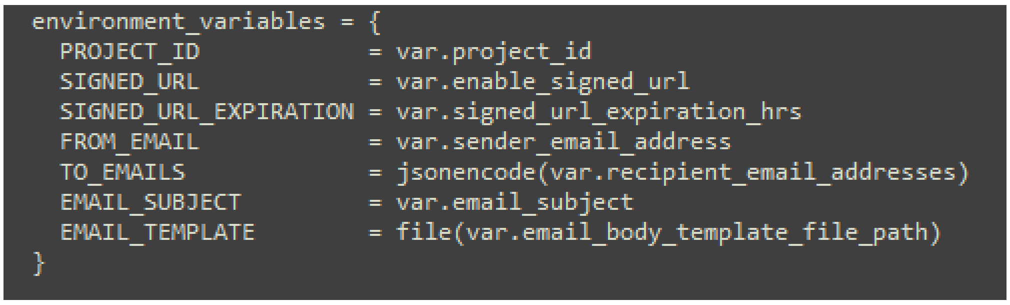


```
# Topic that triggers function 1 that exports file to GCS
resource "google_pubsub_topic" "pubsub_scheduled_query_completed" {
  name    = "scheduled-query-completed"
  project = var.project_id
}

# Topic that triggers function 2 that sends notification
resource "google_pubsub_topic" "pubsub_file_export_completed" {
  name    = "file-upload-completed"
  project = var.project_id
}
```

The google_storage_bucket resource generates a bucket that stores a bucket object which is created by the google_storage_bucket_object resource. This object contains the query results which are exported from BQ as archive files. The google_cloudfunctions_function resource creates the export_query_results_function was built above based on the Python Client Libraries. Inside this resource block, the event_trigger includes the PubSub topic as event_type and the resource where to get this event. The environment_variables argument contains all the information that assigned to the function such as project id, bucket name, object name, compression, destination format, and some. The work flow for implementing the cloud functions and the Pub/Sub to trigger the cloud functions into the send_notification_function same as the export_query_results_function. However, the values are assigned to the environment_variables are different. They are project id, signed url, signed url expiration, from emails, to emails, email subject and email template. The email template body is created as a html file. It can be modified depends on customers requirements.

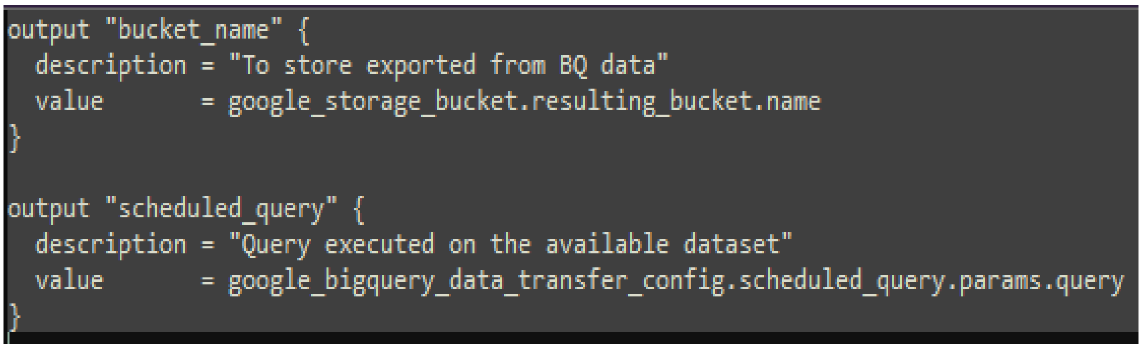


```
resource "google_storage_bucket" "function_bucket_1" {
  name          = "export_query_results_function"
  project       = var.project_id
  location      = var.gcs_location
  force_destroy = true
}

data "archive_file" "init_1" {
  type        = "zip"
  source_dir  = "functions/export_query_results_function"
  output_path = "export_query_results_function_source.zip"
}

resource "google_storage_bucket_object" "archive_1" {
  name   = "export_query_results_function_source.zip"
  bucket = google_storage_bucket.function_bucket_1.name
  source = data.archive_file.init_1.output_path
}

resource "google_cloudfunctions_function" "export_query_results_function" {
  name    = "export_query_results_function"
  project = var.project_id
  runtime = "python37"
  region  = var.gcs_location

  event_trigger {
    event_type = "google.pubsub.topic.publish"
    resource   = google_pubsub_topic.pubsub_scheduled_query_completed.id
  }

  service_account_email = google_service_account.service_account.email
  source_archive_bucket = google_storage_bucket.function_bucket_1.name
  source_archive_object = google_storage_bucket_object.archive_1.name
  entry_point           = "main"
  timeout               = 540

  environment_variables = {
    PROJECT_ID      = var.project_id
    BUCKET_NAME     = google_storage_bucket.resulting_bucket.name
    OBJECT_NAME     = var.export_object_name
    COMPRESSION     = var.export_compression
    DEST_FMT        = var.export_destination_format
    USE_AVRO_TYPES  = var.export_use_avro_logical_types
    FIELD_DELIMITER = var.export_field_delimiter
  }
}
```



```
  environment_variables = {
    PROJECT_ID            = var.project_id
    SIGNED_URL            = var.enable_signed_url
    SIGNED_URL_EXPIRATION = var.signed_url_expiration_hrs
    FROM_EMAIL            = var.sender_email_address
    TO_EMAILS             = jsonencode(var.recipient_email_addresses)
    EMAIL_SUBJECT         = var.email_subject
    EMAIL_TEMPLATE        = file(var.email_body_template_file_path)
  }
```

To deploy this pipeline, the terraform commands as terraform init, terraform plan and terraform deploy will be run. The outputs results will be the bucket name that was used to store the query results files and the scheduled query results that was executed on the billing dataset.



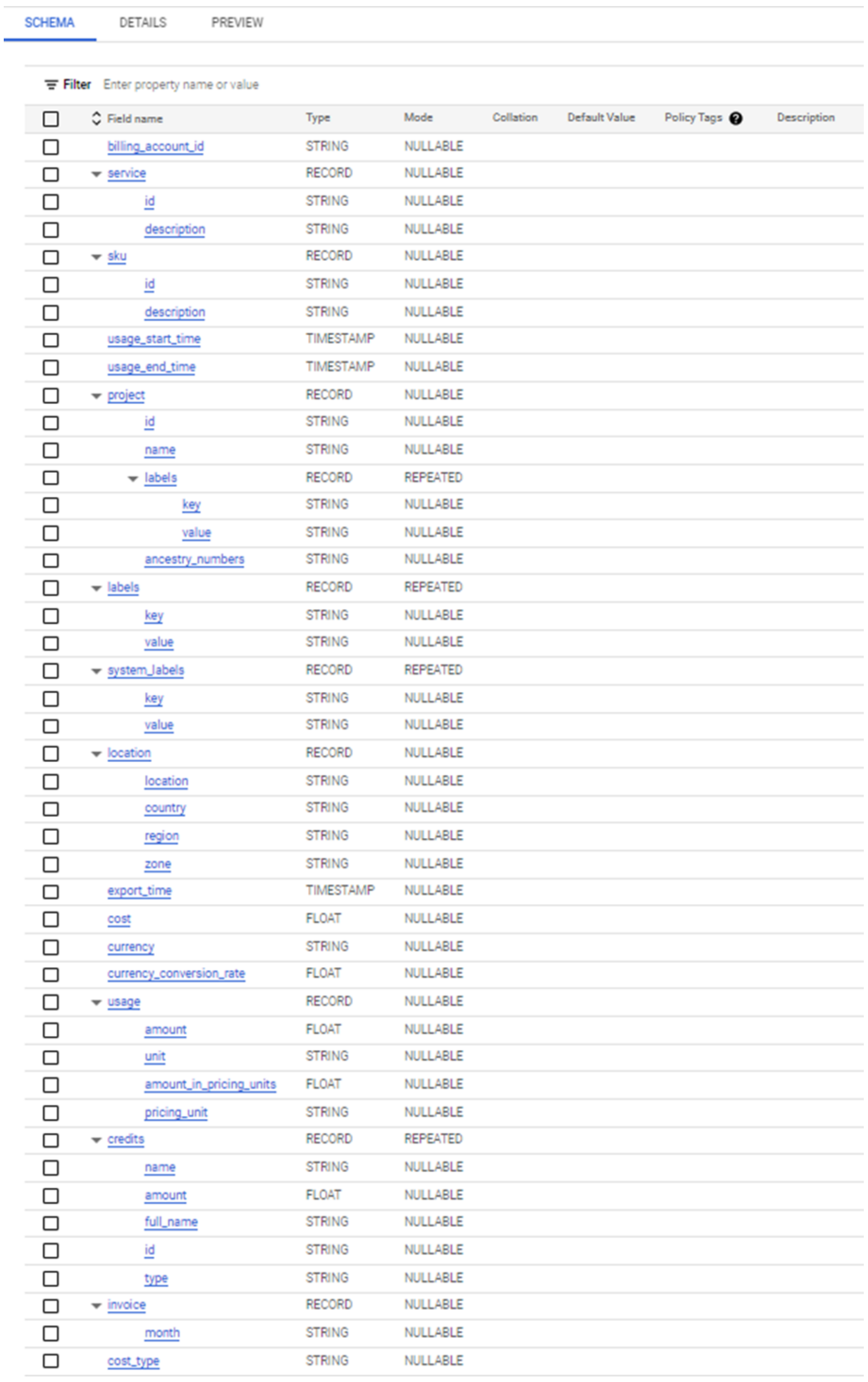
```
output "bucket_name" {
  description = "To store exported from BQ data"
  value       = google_storage_bucket.resulting_bucket.name
}

output "scheduled_query" {
  description = "Query executed on the available dataset"
  value       = google_bigquery_data_transfer_config.scheduled_query.params.query
}
```

5 Results and Analysis

Results and analysis of the solution's implementation are presented in this chapter. This analysis details the testing procedure, the results of the testing, and the implications of those results.

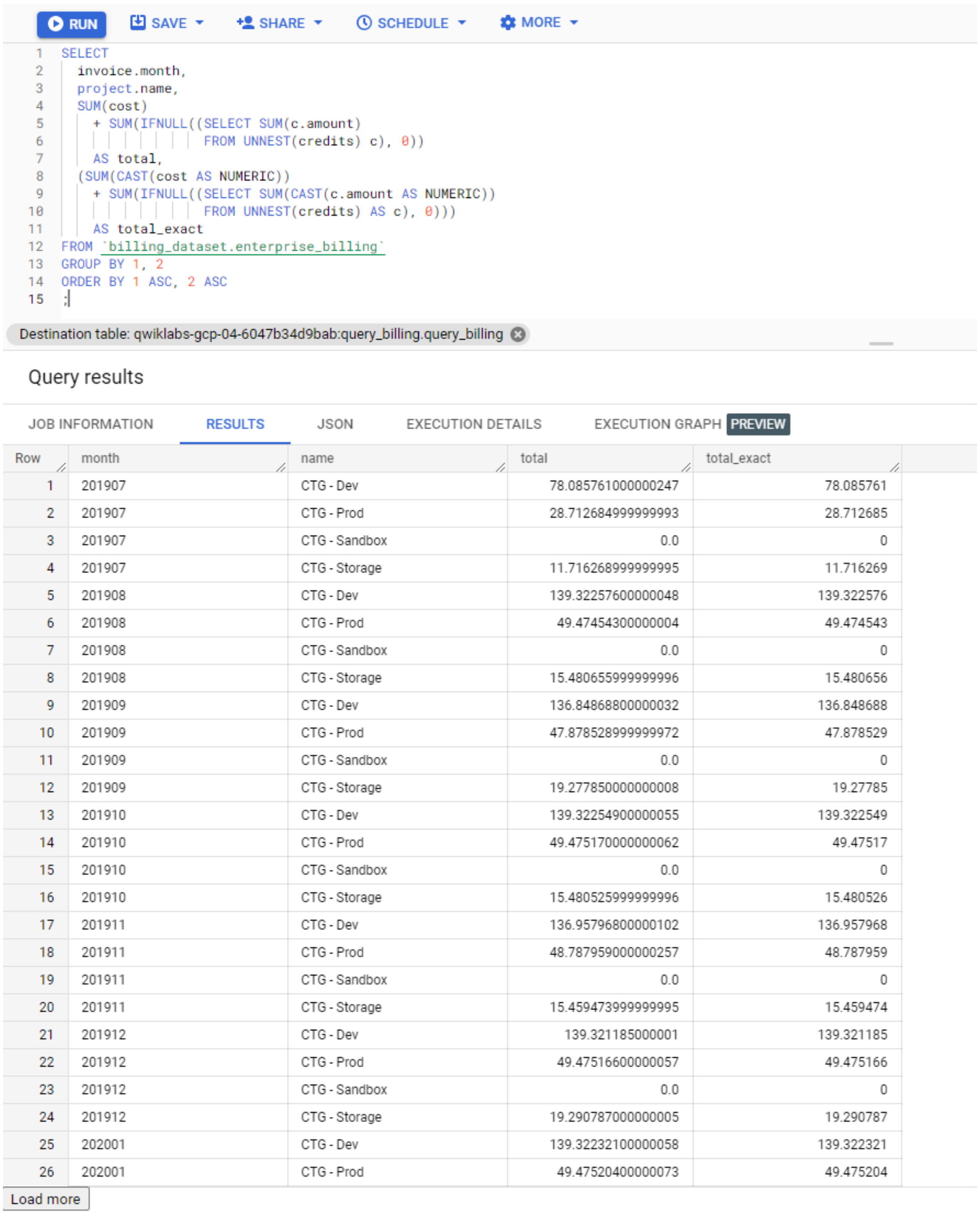
5.1 Results

Standard usage cost data from Google Cloud is exported to BQ, and its structure is described below. Commonly used account details, for example, account ID, projects, SKUs, invoice’s date, labels, locations, cost, usage, credits, adjustments, and currency are all included. The usage costs that are generated by resources such as virtual machines or solid-state drives are not included in the normal cost data. Detailed usage cost data can be enabled if users want to export resource-level cost data to BQ for analysis. Similar to the basic consumption cost data, the exported detailed data also includes all of the same columns and features.



Field name	Type	Mode	Collation	Default Value	Policy Tags	Description
billing_account_id	STRING	NULLABLE				
service	RECORD	NULLABLE				
id	STRING	NULLABLE				
description	STRING	NULLABLE				
sku	RECORD	NULLABLE				
id	STRING	NULLABLE				
description	STRING	NULLABLE				
usage_start_time	TIMESTAMP	NULLABLE				
usage_end_time	TIMESTAMP	NULLABLE				
project	RECORD	NULLABLE				
id	STRING	NULLABLE				
name	STRING	NULLABLE				
labels	RECORD	REPEATED				
key	STRING	NULLABLE				
value	STRING	NULLABLE				
ancestry_numbers	STRING	NULLABLE				
labels	RECORD	REPEATED				
key	STRING	NULLABLE				
value	STRING	NULLABLE				
system_labels	RECORD	REPEATED				
key	STRING	NULLABLE				
value	STRING	NULLABLE				
location	RECORD	NULLABLE				
location	STRING	NULLABLE				
country	STRING	NULLABLE				
region	STRING	NULLABLE				
zone	STRING	NULLABLE				
export_time	TIMESTAMP	NULLABLE				
cost	FLOAT	NULLABLE				
currency	STRING	NULLABLE				
currency_conversion_rate	FLOAT	NULLABLE				
usage	RECORD	NULLABLE				
amount	FLOAT	NULLABLE				
unit	STRING	NULLABLE				
amount_in_pricing_units	FLOAT	NULLABLE				
pricing_unit	STRING	NULLABLE				
credits	RECORD	REPEATED				
name	STRING	NULLABLE				
amount	FLOAT	NULLABLE				
full_name	STRING	NULLABLE				
id	STRING	NULLABLE				
type	STRING	NULLABLE				
invoice	RECORD	NULLABLE				
month	STRING	NULLABLE				
cost_type	STRING	NULLABLE				

The monthly invoice total, including all fees, taxes, and other charges, as well as any modifications or rounding errors, is displayed in the following query for each project.name. Costs that are not related to a project-level item are aggregated for the month under the name null.



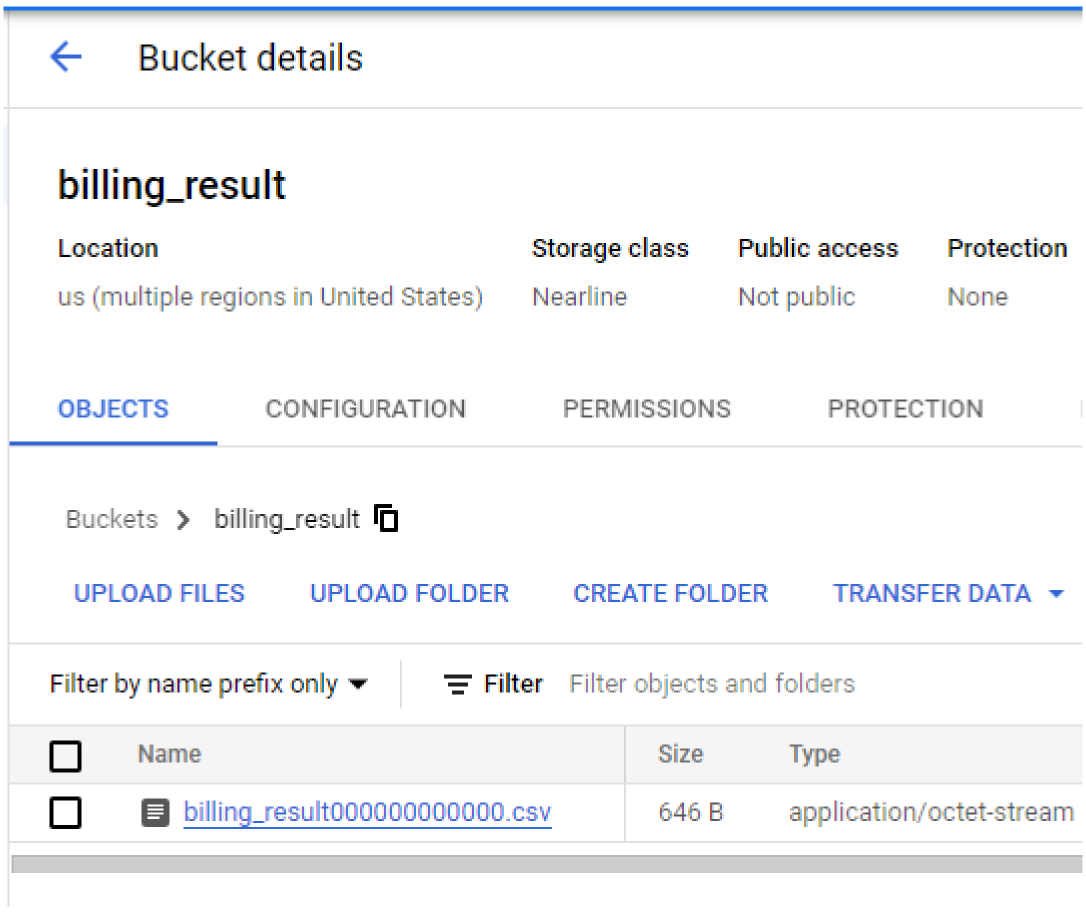
```sql
SELECT
  invoice.month,
  project.name,
  SUM(cost)
    + SUM(IFNULL((SELECT SUM(c.amount)
                  FROM UNNEST(credits) c), 0))
    AS total,
  (SUM(CAST(cost AS NUMERIC))
    + SUM(IFNULL((SELECT SUM(CAST(c.amount AS NUMERIC))
                  FROM UNNEST(credits) AS c), 0)))
    AS total_exact
FROM `billing_dataset.enterprise_billing`
GROUP BY 1, 2
ORDER BY 1 ASC, 2 ASC
;
```

Destination table: qwiklabs-gcp-04-6047b34d9bab:query_billing.query_billing

Query results

Row	month	name	total	total_exact
1	201907	CTG - Dev	78.085761000000247	78.085761
2	201907	CTG - Prod	28.712684999999993	28.712685
3	201907	CTG - Sandbox	0.0	0
4	201907	CTG - Storage	11.716268999999995	11.716269
5	201908	CTG - Dev	139.32257600000048	139.322576
6	201908	CTG - Prod	49.47454300000004	49.474543
7	201908	CTG - Sandbox	0.0	0
8	201908	CTG - Storage	15.480655999999996	15.480656
9	201909	CTG - Dev	136.84868800000032	136.848688
10	201909	CTG - Prod	47.878528999999972	47.878529
11	201909	CTG - Sandbox	0.0	0
12	201909	CTG - Storage	19.277850000000008	19.27785
13	201910	CTG - Dev	139.32254900000055	139.322549
14	201910	CTG - Prod	49.475170000000062	49.47517
15	201910	CTG - Sandbox	0.0	0
16	201910	CTG - Storage	15.480525999999996	15.480526
17	201911	CTG - Dev	136.95796800000102	136.957968
18	201911	CTG - Prod	48.787959000000257	48.787959
19	201911	CTG - Sandbox	0.0	0
20	201911	CTG - Storage	15.459473999999995	15.459474
21	201912	CTG - Dev	139.321185000001	139.321185
22	201912	CTG - Prod	49.47516600000057	49.475166
23	201912	CTG - Sandbox	0.0	0
24	201912	CTG - Storage	19.290787000000005	19.290787
25	202001	CTG - Dev	139.32232100000058	139.322321
26	202001	CTG - Prod	49.47520400000073	49.475204

The query results table from BQ was promptly exported to the cloud storage as a single CSV file when the Pub/Sub triggered the first cloud function, namely the export_query_results_function. However, if the file size exceeds 1 GB, it will be split into several smaller files.



Bucket details

billing_result

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Nearline	Not public	None

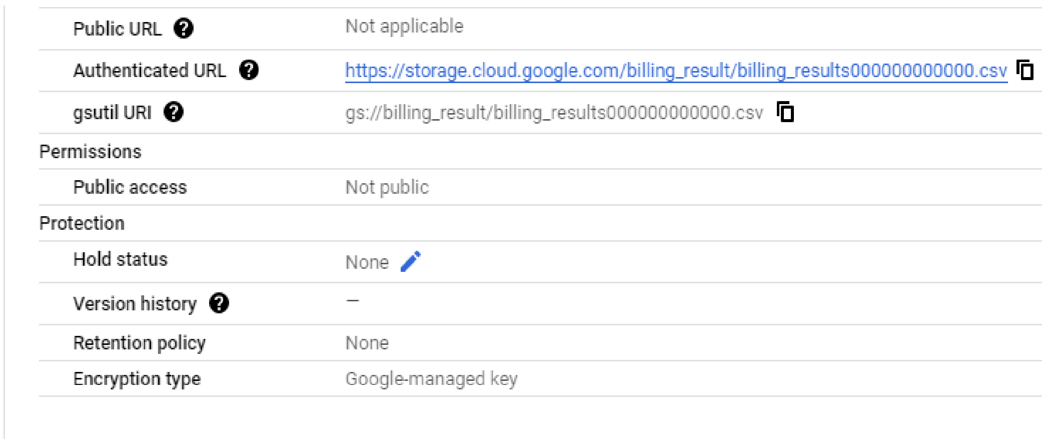
OBJECTS CONFIGURATION PERMISSIONS PROTECTION

Buckets > billing_result

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DATA

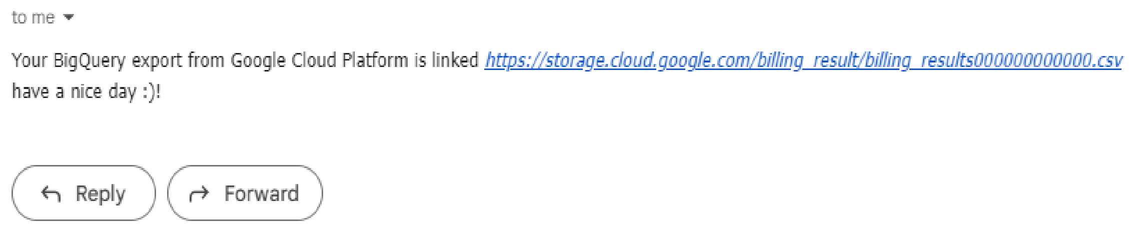
Filter by name prefix only Filter Filter objects and folders

Name	Size	Type
billing_result000000000000.csv	646 B	application/octet-stream



Public URL	Not applicable
Authenticated URL	https://storage.cloud.google.com/billing_result/billing_results000000000000.csv
gsutil URI	gs://billing_result/billing_results000000000000.csv
Permissions	
Public access	Not public
Protection	
Hold status	None
Version history	–
Retention policy	None
Encryption type	Google-managed key

Once the second Pub/Sub triggered the second cloud function, namely send_notification_function, the email would be sent to the customers with the GCS signed URL via the SendGrid API.


to me

Your BigQuery export from Google Cloud Platform is linked https://storage.cloud.google.com/billing_result/billing_results000000000000.csv
have a nice day :)!

5.2 Analysis

In a growing cloud environment, it might be difficult to get a handle on cloud costs. When comparing cost tracking for a small project hosting a single web server to that of a huge corporation with hundreds of projects and thousands of individuals working at varying levels in the cloud, there is a significant gap. In order to make sure it is spending money on the correct things and to plan for the future, it may be quite helpful for a business to keep tabs on and analyse the resources that go into calculating its expenditures. There are a number of reasons to activate the billing export as soon as possible, but one of the most crucial is that the information cannot be retrieved from the past. As a result, customers will be out of luck if they want to look at the billing information from the previous month and they forgot to turn on export. Because of this, users should turn on the export as soon as they set up their billing account.

Use the built-in reporting tools to quickly acquire answers and gain insight into business costs. It is a matter of magnitude and detail that separates these two cases. When teams and companies grow in size, it becomes more complicated to assign financial responsibility. The additional granularity is especially important for organizations that use a chargeback model (in which a central group pays the bill for all resources and then provides cost data to individual teams or groups to hold them accountable for their usage).

Using the exports to their full potential is likely to necessitate familiarity with SQL or employment in a data analysis capacity. However, users can still get a handle on the billing data by enabling the export and then setting up visualization tools with minimal effort. People typically believe they do not require this level of detail at first but later come to wish they had it as their cloud usage expands.

The example queries and schema documentation are the most helpful parts of this site. These materials are comprehensive guides that will help users understand the data and get started. Also, remember that it is preferable to begin modestly and gradually progress. It is extremely difficult and time-consuming to develop a comprehensive cost reporting solution for an entire organization from scratch. Learn to walk by breaking down costs by project and month. Then, learn to run by setting up anomaly detection and alerting teams when resources under their control are causing unexpected cost overruns.

6 Discussions and Conclusions

6.1 Conclusions

BQ is a service available on Google Cloud Platform for the purposes of data warehousing, analytics, and machine learning, as well as the creation, management, distribution, and querying of data. BQ can be used for cloud-based parallel computing for large-scale data analysis.

Using GCP Billing Export to Google BQ, users can export comprehensive Google Cloud billing information into a BQ dataset (such as Utilization, Projected Costs, and Unit Prices). Cloud Billing information may now be viewed in Google Data Studio or analysed in Google BQ for further depth. Information can be exported to a JSON file via the GCP Billing Export method, which can then be read by other programs.

Finally, BQ is an excellent complementary tool for analysing the GCP billing data. To achieve the desired outcomes, it is crucial to remember that all relevant procedures must be carried out carefully. BQ should be used with complete understanding to avoid wasting money on unnecessary data and operations. It is important to remember that there are still several restrictions to consider.

6.2 Limitations and Improvements

The maximum amount of data that may be exported from BQ in a single file is one gigabyte (GB). If the results of the query are more than one gigabyte, users will need to export the data to numerous files in GCS; unfortunately, this solution does not provide that functionality. Another option would be to make use of GCS Compose in order to affix together a number of separate items into a single file that could then be sent.

Signed URLs provide a potential danger for the exfiltration of sensitive data. Take into consideration the potential safety issues that might arise from the transfer of data over a signed URL.

If the use case users have does not satisfy the requirements outlined above, users may want to consider using a Cloud Composer process as an alternative to putting the pipeline into action. This solution may also be accomplished with a scheduled Apps Script by making use of the BQ Service and exporting data to a Google Sheet. If users are a customer of GSuite, users can take advantage of this feature.

It is not feasible to export data that is nested or repeated when using the CSV format. Exporting data in Avro, JSON, and Parquet all enable nested and repeated data, respectively. It indicates that the query that is used in a scheduled query should not build the dataset with the nested fields if a CSV file is going to be utilized.

References

1 Divya R, Jayanthi V. A SURVEY ON CLOUD OPTIMIZATION SYSTEMS.

2 Naidu S, Tigani J. Google BigQuery Analytics. John Wiley & Sons; 2014 May 21.

3 Google Cloud. BQ table. [Online]. Google Cloud; 2022 [cited 2022 Jun 18]. Available from: https://cloud.google.com/BQ/docs/tables-intro.

4 Mucchetti M. Managing BigQuery Costs. In BigQuery for Data Warehousing 2020 (pp. 61-71). Apress, Berkeley, CA.

5 Mucchetti M. Loading Data into the Warehouse. In BigQuery for Data Warehousing 2020 (pp. 75-103). Apress, Berkeley, CA.

6 Google Cloud. BQ writing results. [Online]. Google Cloud; 2022 [cited 2022 Jun 18]. Available from: https://cloud.google.com/BQ/docs/writing-results.

7 Google Cloud. BQ data analytics use case example. [Online]. The cloud girl; 2022 [cited 2022 Jun 18]. Available from: https://thecloudgirl.dev/BQ.html.

8 Bisong E. Google Cloud Storage (GCS). In Building Machine Learning and Deep Learning Models on Google Cloud Platform 2019 (pp. 25-33). Apress, Berkeley, CA.

9 Krishnan SP, Gonzalez JL. Cloud Storage. In Building Your Next Big Thing with Google Cloud Platform 2015 (pp. 185-210). Apress, Berkeley, CA.

10 Antu AD, Kumar A, Kelley R, Xie B. Comparative Analysis of Cloud Storage Options for Diverse Application Requirements. In International Conference on Cloud Computing 2021 Dec 10 (pp. 75-96). Springer, Cham.

11 Google Cloud. Google cloud storage use case example. [Online]. The cloud girl; 2022 [cited 2022 Jun 18]. Available from: https://thecloudgirl.dev/CloudStorage.html.

12 Sullivan, Dan. Computing with Cloud Functions. 2019: 225-240.

13 Google Cloud. Google cloud functions use case example. [Online]. The cloud girl; 2022 [cited 2022 Jun 18]. Available from: https://thecloudgirl.dev/CloudFunctions.html.

14 Kumar M. Google cloud platform: a powerful big data analytics cloud platform. Int J Res Appl Sci Eng Technol. 2016 Nov;4(11):387-92.

15 Google Cloud. Google cloud pub/sub message flows. [Online]. The cloud girl; 2022 [cited 2022 Jun 18]. Available from: https://thecloudgirl.dev/pubsub.html.

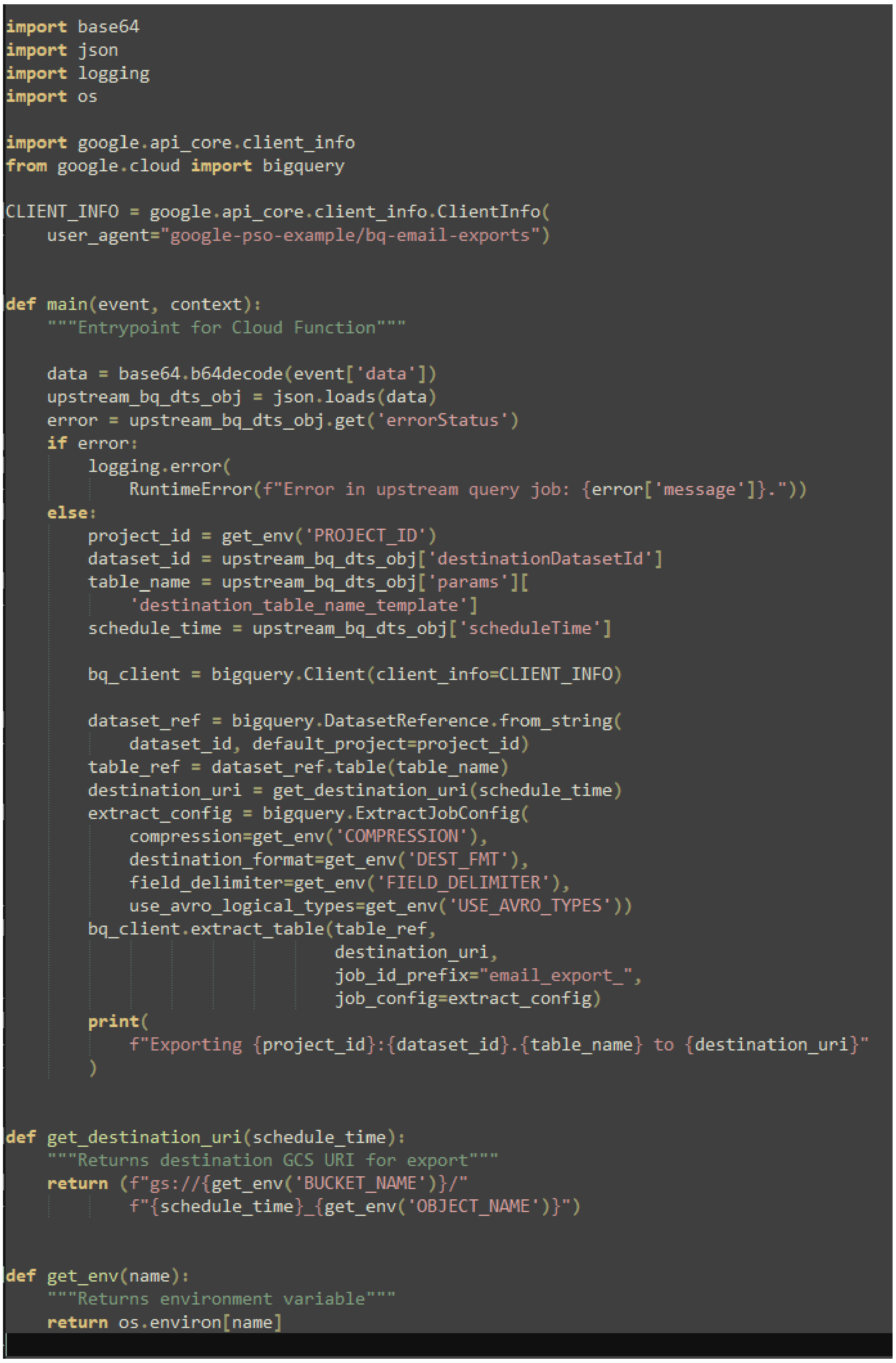
16 Krishnan SP, Gonzalez JL. Google cloud pub/sub. In Building Your Next Big Thing with Google Cloud Platform 2015 (pp. 277-292). Apress, Berkeley, CA.

17 Zadka M. Terraform. In DevOps in Python 2022 (pp. 225-230). Apress, Berkeley, CA.

18 Cukier D. DevOps patterns to scale web applications using cloud services. InProceedings of the 2013 companion publication for conference on Systems, programming, & applications: software for humanity 2013 Oct 26 (pp. 143-152).

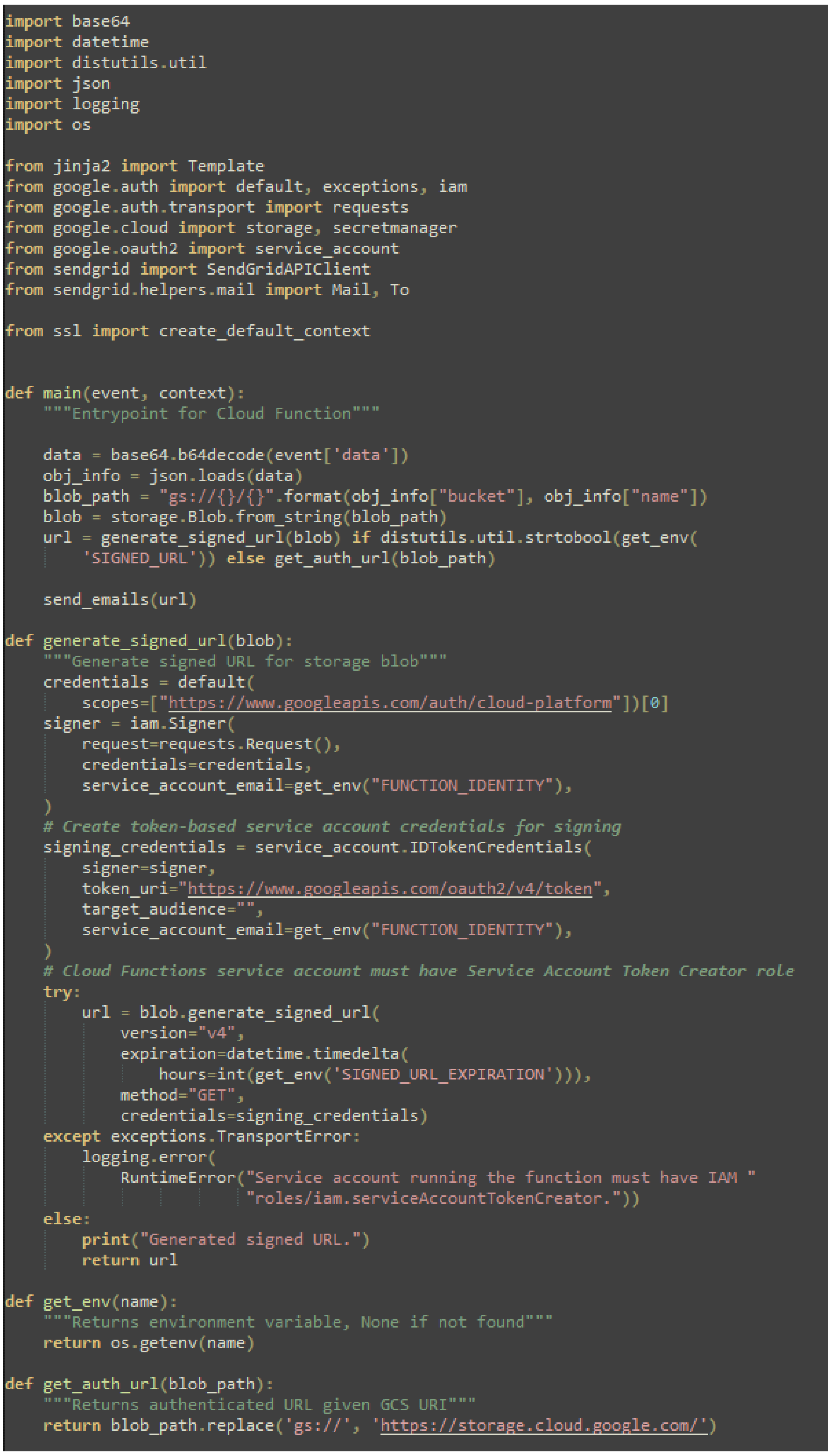
19 Kelly S. What Is Python? In Python, PyGame and Raspberry Pi Game Development 2016 (pp. 3-5). Apress, Berkeley, CA.

Export_query_results_function

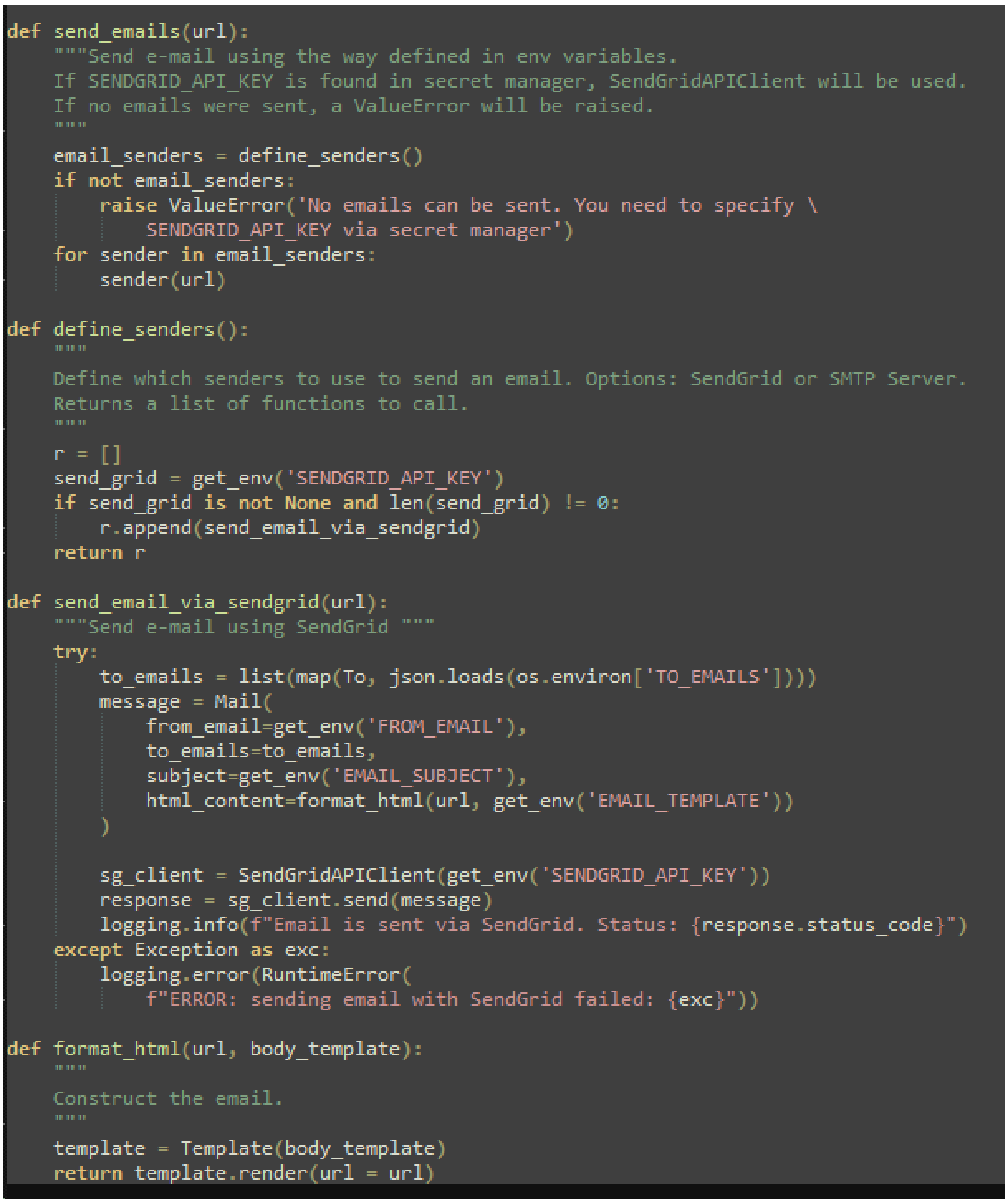


```python
import base64
import json
import logging
import os

import google.api_core.client_info
from google.cloud import bigquery

CLIENT_INFO = google.api_core.client_info.ClientInfo(
    user_agent="google-pso-example/bq-email-exports")


def main(event, context):
    """Entrypoint for Cloud Function"""

    data = base64.b64decode(event['data'])
    upstream_bq_dts_obj = json.loads(data)
    error = upstream_bq_dts_obj.get('errorStatus')
    if error:
        logging.error(
            RuntimeError(f"Error in upstream query job: {error['message']}."))
    else:
        project_id = get_env('PROJECT_ID')
        dataset_id = upstream_bq_dts_obj['destinationDatasetId']
        table_name = upstream_bq_dts_obj['params'][
            'destination_table_name_template']
        schedule_time = upstream_bq_dts_obj['scheduleTime']

        bq_client = bigquery.Client(client_info=CLIENT_INFO)

        dataset_ref = bigquery.DatasetReference.from_string(
            dataset_id, default_project=project_id)
        table_ref = dataset_ref.table(table_name)
        destination_uri = get_destination_uri(schedule_time)
        extract_config = bigquery.ExtractJobConfig(
            compression=get_env('COMPRESSION'),
            destination_format=get_env('DEST_FMT'),
            field_delimiter=get_env('FIELD_DELIMITER'),
            use_avro_logical_types=get_env('USE_AVRO_TYPES'))
        bq_client.extract_table(table_ref,
                                destination_uri,
                                job_id_prefix="email_export_",
                                job_config=extract_config)
        print(
            f"Exporting {project_id}:{dataset_id}.{table_name} to {destination_uri}"
        )


def get_destination_uri(schedule_time):
    """Returns destination GCS URI for export"""
    return (f"gs://{get_env('BUCKET_NAME')}/"
            f"{schedule_time}_{get_env('OBJECT_NAME')}")


def get_env(name):
    """Returns environment variable"""
    return os.environ[name]
```

Send_notification_function



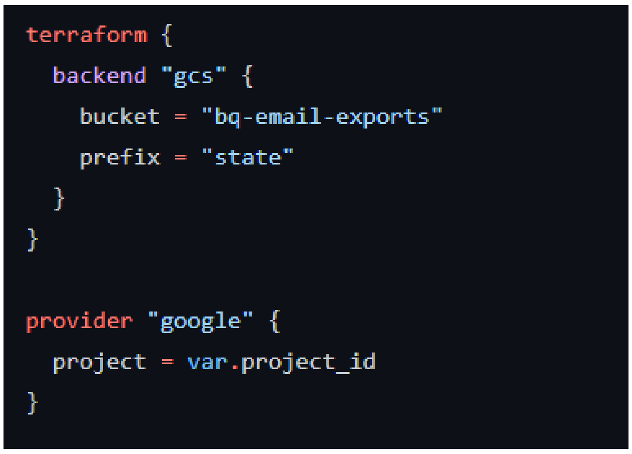
```python
import base64
import datetime
import distutils.util
import json
import logging
import os

from jinja2 import Template
from google.auth import default, exceptions, iam
from google.auth.transport import requests
from google.cloud import storage, secretmanager
from google.oauth2 import service_account
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail, To

from ssl import create_default_context


def main(event, context):
    """Entrypoint for Cloud Function"""

    data = base64.b64decode(event['data'])
    obj_info = json.loads(data)
    blob_path = "gs://{}/{}".format(obj_info["bucket"], obj_info["name"])
    blob = storage.Blob.from_string(blob_path)
    url = generate_signed_url(blob) if distutils.util.strtobool(get_env(
        'SIGNED_URL')) else get_auth_url(blob_path)

    send_emails(url)

def generate_signed_url(blob):
    """Generate signed URL for storage blob"""
    credentials = default(
        scopes=["https://www.googleapis.com/auth/cloud-platform"])[0]
    signer = iam.Signer(
        request=requests.Request(),
        credentials=credentials,
        service_account_email=get_env("FUNCTION_IDENTITY"),
    )
    # Create token-based service account credentials for signing
    signing_credentials = service_account.IDTokenCredentials(
        signer=signer,
        token_uri="https://www.googleapis.com/oauth2/v4/token",
        target_audience="",
        service_account_email=get_env("FUNCTION_IDENTITY"),
    )
    # Cloud Functions service account must have Service Account Token Creator role
    try:
        url = blob.generate_signed_url(
            version="v4",
            expiration=datetime.timedelta(
                hours=int(get_env('SIGNED_URL_EXPIRATION'))),
            method="GET",
            credentials=signing_credentials)
    except exceptions.TransportError:
        logging.error(
            RuntimeError("Service account running the function must have IAM "
                         "roles/iam.serviceAccountTokenCreator."))
    else:
        print("Generated signed URL.")
        return url

def get_env(name):
    """Returns environment variable, None if not found"""
    return os.getenv(name)

def get_auth_url(blob_path):
    """Returns authenticated URL given GCS URI"""
    return blob_path.replace('gs://', 'https://storage.cloud.google.com/')
```



```python
def send_emails(url):
    """Send e-mail using the way defined in env variables.
    If SENDGRID_API_KEY is found in secret manager, SendGridAPIClient will be used.
    If no emails were sent, a ValueError will be raised.
    """
    email_senders = define_senders()
    if not email_senders:
        raise ValueError('No emails can be sent. You need to specify \
            SENDGRID_API_KEY via secret manager')
    for sender in email_senders:
        sender(url)

def define_senders():
    """
    Define which senders to use to send an email. Options: SendGrid or SMTP Server.
    Returns a list of functions to call.
    """
    r = []
    send_grid = get_env('SENDGRID_API_KEY')
    if send_grid is not None and len(send_grid) != 0:
        r.append(send_email_via_sendgrid)
    return r

def send_email_via_sendgrid(url):
    """Send e-mail using SendGrid """
    try:
        to_emails = list(map(To, json.loads(os.environ['TO_EMAILS'])))
        message = Mail(
            from_email=get_env('FROM_EMAIL'),
            to_emails=to_emails,
            subject=get_env('EMAIL_SUBJECT'),
            html_content=format_html(url, get_env('EMAIL_TEMPLATE'))
        )

        sg_client = SendGridAPIClient(get_env('SENDGRID_API_KEY'))
        response = sg_client.send(message)
        logging.info(f"Email is sent via SendGrid. Status: {response.status_code}")
    except Exception as exc:
        logging.error(RuntimeError(
            f"ERROR: sending email with SendGrid failed: {exc}"))

def format_html(url, body_template):
    """
    Construct the email.
    """
    template = Template(body_template)
    return template.render(url = url)
```

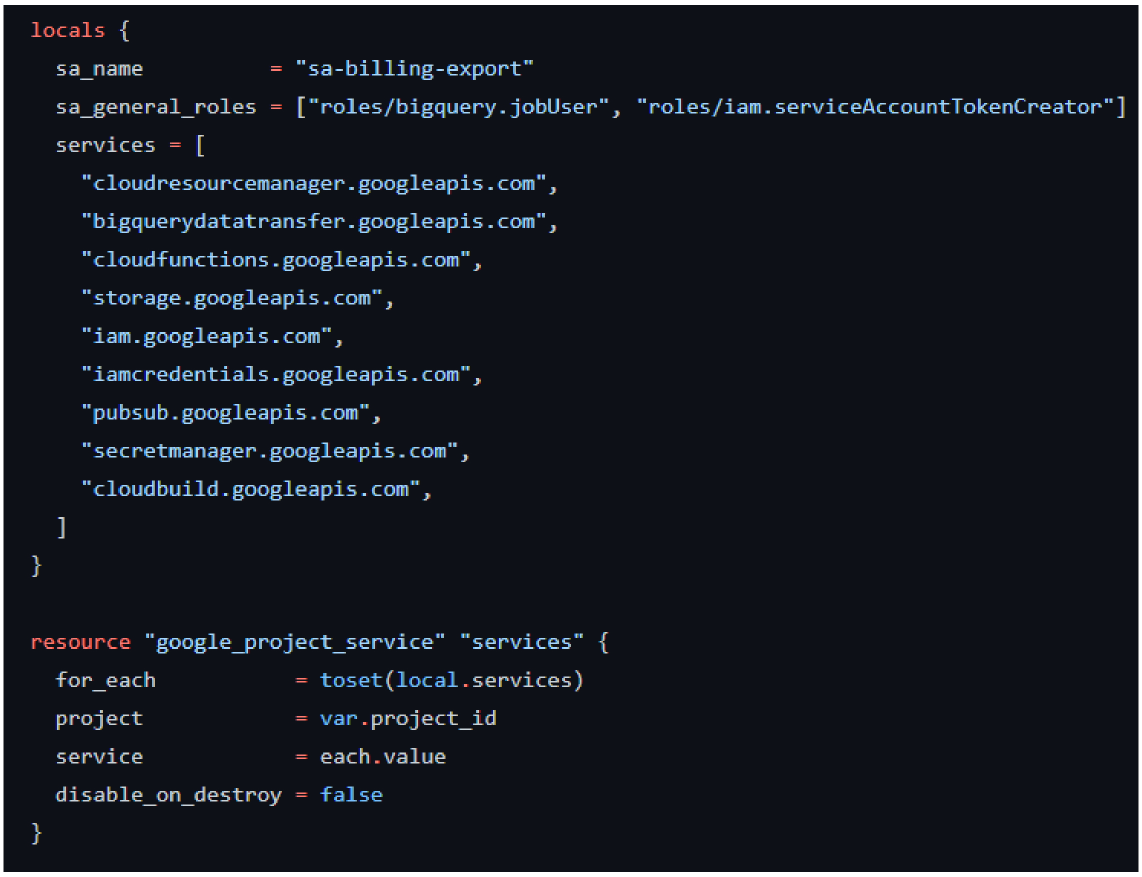
Terraform solution

Provider



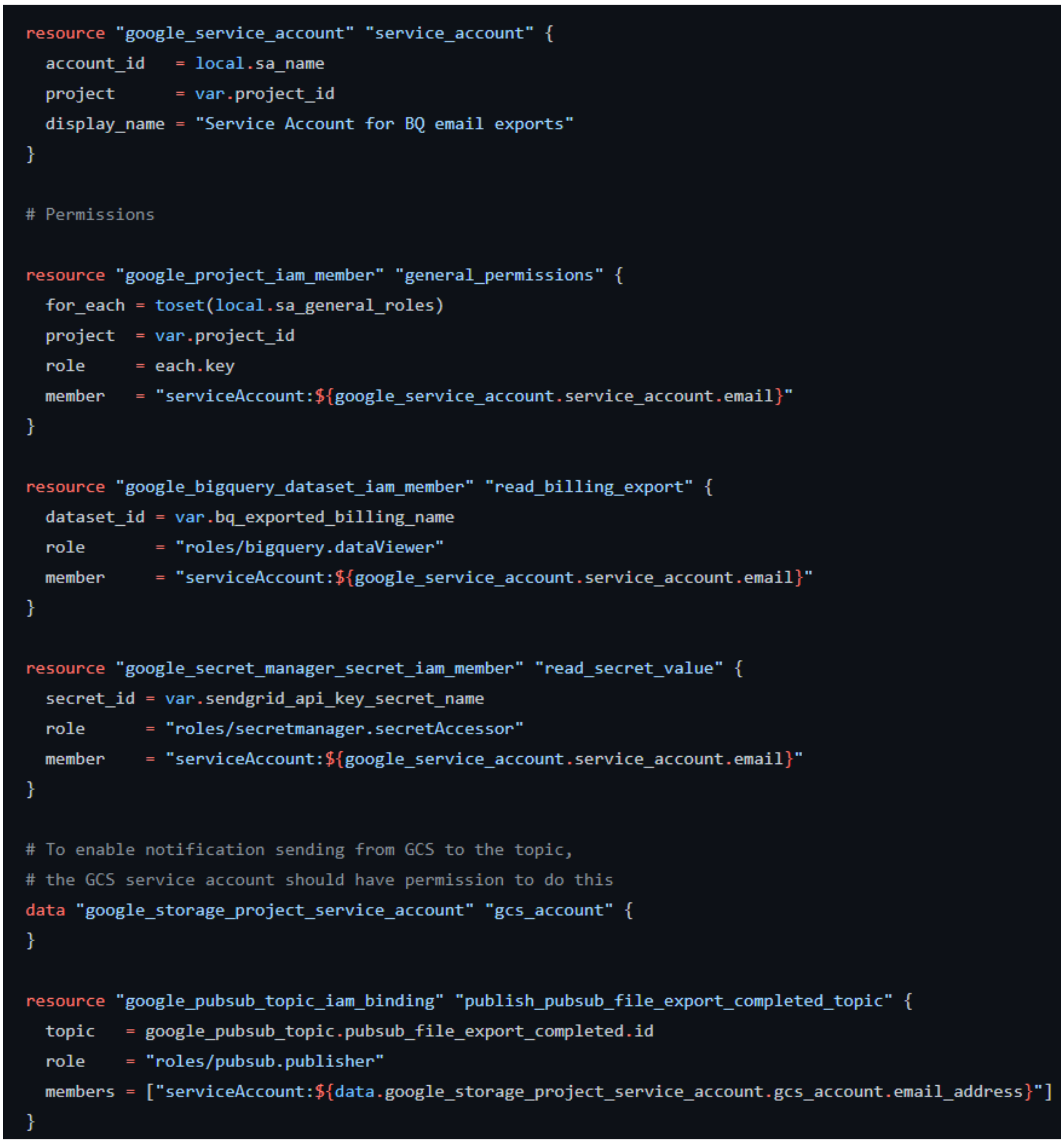
```
terraform {
  backend "gcs" {
    bucket = "bq-email-exports"
    prefix = "state"
  }
}

provider "google" {
  project = var.project_id
}
```

Enable APIs

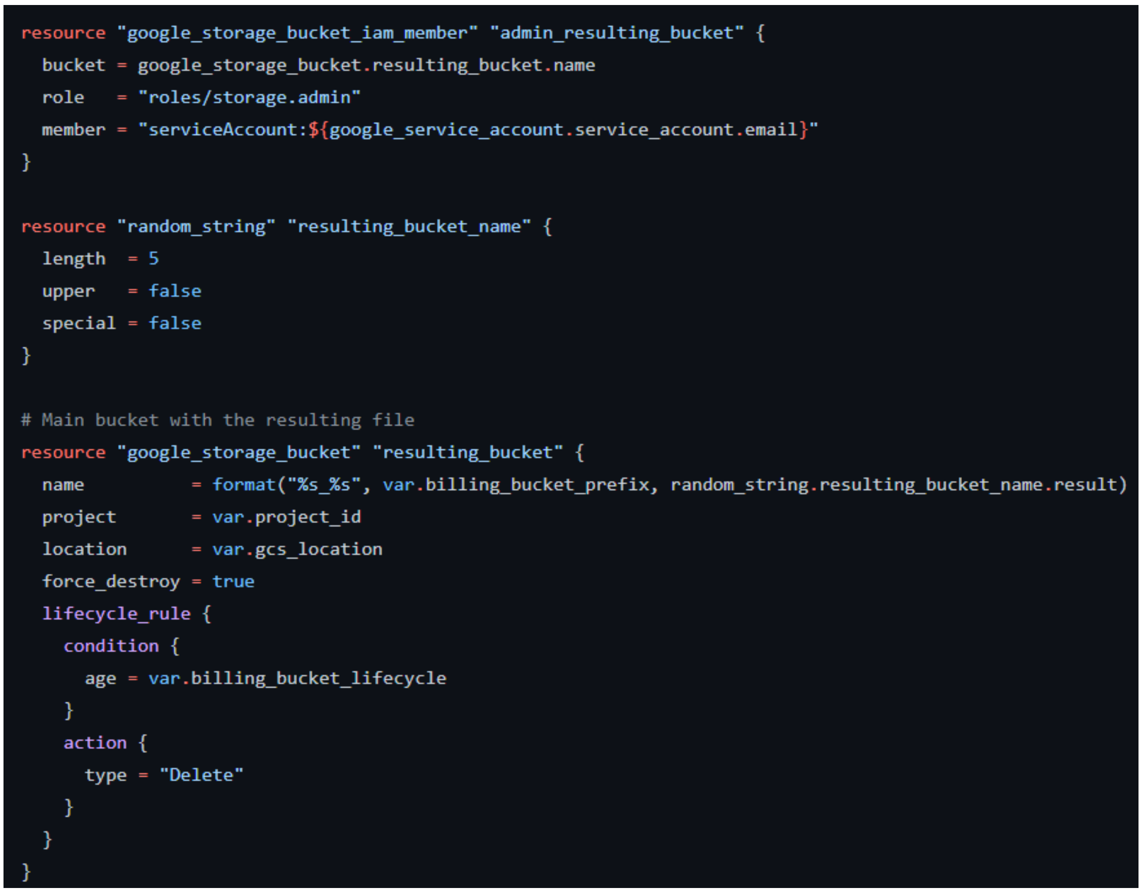


```
locals {
  sa_name          = "sa-billing-export"
  sa_general_roles = ["roles/bigquery.jobUser", "roles/iam.serviceAccountTokenCreator"]
  services = [
    "cloudresourcemanager.googleapis.com",
    "bigquerydatatransfer.googleapis.com",
    "cloudfunctions.googleapis.com",
    "storage.googleapis.com",
    "iam.googleapis.com",
    "iamcredentials.googleapis.com",
    "pubsub.googleapis.com",
    "secretmanager.googleapis.com",
    "cloudbuild.googleapis.com",
  ]
}

resource "google_project_service" "services" {
  for_each           = toset(local.services)
  project            = var.project_id
  service            = each.value
  disable_on_destroy = false
}
```

Creating Service Account and Permissions



```
resource "google_service_account" "service_account" {
  account_id   = local.sa_name
  project      = var.project_id
  display_name = "Service Account for BQ email exports"
}

# Permissions

resource "google_project_iam_member" "general_permissions" {
  for_each = toset(local.sa_general_roles)
  project  = var.project_id
  role     = each.key
  member   = "serviceAccount:${google_service_account.service_account.email}"
}

resource "google_bigquery_dataset_iam_member" "read_billing_export" {
  dataset_id = var.bq_exported_billing_name
  role       = "roles/bigquery.dataViewer"
  member     = "serviceAccount:${google_service_account.service_account.email}"
}

resource "google_secret_manager_secret_iam_member" "read_secret_value" {
  secret_id = var.sendgrid_api_key_secret_name
  role      = "roles/secretmanager.secretAccessor"
  member    = "serviceAccount:${google_service_account.service_account.email}"
}

# To enable notification sending from GCS to the topic,
# the GCS service account should have permission to do this
data "google_storage_project_service_account" "gcs_account" {
}

resource "google_pubsub_topic_iam_binding" "publish_pubsub_file_export_completed_topic" {
  topic   = google_pubsub_topic.pubsub_file_export_completed.id
  role    = "roles/pubsub.publisher"
  members = ["serviceAccount:${data.google_storage_project_service_account.gcs_account.email_address}"]
}
```



```
resource "google_storage_bucket_iam_member" "admin_resulting_bucket" {
  bucket = google_storage_bucket.resulting_bucket.name
  role   = "roles/storage.admin"
  member = "serviceAccount:${google_service_account.service_account.email}"
}

resource "random_string" "resulting_bucket_name" {
  length  = 5
  upper   = false
  special = false
}

# Main bucket with the resulting file
resource "google_storage_bucket" "resulting_bucket" {
  name          = format("%s_%s", var.billing_bucket_prefix, random_string.resulting_bucket_name.result)
  project       = var.project_id
  location      = var.gcs_location
  force_destroy = true
  lifecycle_rule {
    condition {
      age = var.billing_bucket_lifecycle
    }
    action {
      type = "Delete"
    }
  }
}
```