Date of publication SEP-30, 2022, date of current version FEB-04, 2022.
www.computingonline.net / computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.21.3.2694

A Cloud Pub/Sub Architecture to Integrate Google Big Query with Elasticsearch using Cloud Functions

SERGIO LAUREANO GUTIÉRREZ, YASIEL PÉREZ VERA
Escuela Profesional de Ingeniería de Sistemas, Universidad Nacional de San Agustín, Arequipa-Perú
(E-mail: slaureano@unsa.edu.pe, yperezv@unsa.edu.pe)

Corresponding author: Sergio Laureano Gutiérrez (e-mail: slaureano@unsa.edu.pe).

ABSTRACT In recent years, the need for analytics on large volumes of data has become increasingly important. It turns out to be extremely useful in making strategic decisions about different applications. In this way, appropriate mechanisms must be designed to carry out data processing and integration with different platforms to take advantage of their best features. In this work, an architecture that works on cloud services is shown to migrate data stored in Big Query to an analytics engine such as Elasticsearch and take advantage of its potential in query, insert and display operations. This is accomplished through the use of Cloud Functions and Pub / Sub. The integration of these platforms through the proposed architecture showed 100% effectiveness when transferring data to another, maintaining an insertion rate of 4,138.30 documents per second, demonstrating its robustness, efficiency, and versatility when performing a data migration. This pretends to establish an architecture solution when it comes about handling a large amount of data as in the real world.

KEYWORDS Cloud Computing Architecture; Cloud Function; Cloud Service; Serverless; Pub/Sub.

I. INTRODUCTION

IN this digital age, large amounts of data are being generated exponentially daily. Still, the real challenge is to analyze and extract value from them since we are not taking advantage of even a small percentage of what we have at our disposal [1]. We are in a world that is being driven by data. These should allow companies to make better decisions to improve in all aspects [2].

According to the data published in a report [3] by the Seagate company and the IDC consultancy, by 2025, more than 175 Zeta Bytes of data will have been created in the world, an amount that will be ten times higher than that registered in 2016. The study also ensures that by 2025 about 20% of global data will be critical to our daily lives and reveals that the connected persons worldwide will interact with connected devices an average of close to 4,800 times daily [3].

Several essential elements must be taken into account to obtain valuable information. Without these, any effort to deal with large volumes of data is almost certainly not going to work [1]. We mainly faced storage problems in the past, but now, storing large amounts of information is no longer particularly difficult [4]. Although the storage of large amounts of data is still expensive on many occasions, to solve this problem, we have cloud solutions [5]. A good infrastructure allows us to store and maintain data, but it is very little use without the right tools to access it [6]. The tools for managing large volumes of data are undergoing rapid evolution and must be constantly attentive and updated [7].

Among these tools for handling large volumes of data is Big Query, an enterprise data warehouse solution with support for analysis for vast volumes of data at the scale of Petabytes [8]. The goal of placing the data in Google Big Query enables faster analysis on larger volumes of data. Making data accessible has become paramount. Therefore, making data available as quickly as possible is essential [9]. Traditional data integration solutions are often complex to install, configure, maintain, and develop data flow [10].

Elasticsearch is a search engine for querying large volumes of complex data [11]. Its main characteristic is that it allows the data to be indexed. The answers to the executed queries are fast, being able to analyze the data much more efficiently. It allows large amounts of data processing and seeing their evolution in real-time [12]. In addition, it provides graphs that help to understand more easily the information obtained [13]. One of the benefits of this tool is that it can be expanded with Elastic Stack, a suite of products that enhance the capabilities of Elasticsearch [14].

Both Big Query and Elasticsearch aim to handle large volumes of data. The storage power of Big Query is extremely useful for storing data [15]. On the other hand, the data aggregation capabilities, the queries in DSL format, and the

dynamic data visualization module of Elasticsearch bring significant added value to the big data solutions [16]. The interoperability of both systems can be merged to make more robust applications that handle and display large volumes of information optimally.

This paper presents a cloud architecture that integrates the Big Query non-relational database with the Elasticsearch distributed analysis and analytics engine. It is intended to use cloud tools to interoperability systems such as Cloud Functions on the Pub/Sub architecture to achieve this objective. This paper is structured as follows. Section 2 reviews the related works associated with architectures for system integrations. Subsequently, Section 3 revises the main characteristics of cloud architectures and non-relational databases. Section 4 introduces the cloud architecture proposed to integrate Big Query with Elasticsearch using Cloud Functions. Section 5 shows the results of the proposal's implementation, and Section 6 concludes the paper.

II. RELATED WORKS

From the perspective of the paper "Analyzing Open Source GitHub Repositories Towards Technology Acceptance Model" [17], a proposal for analyzing public data from GitHub repositories available to the public in Big Query was presented. The article investigated the correlations and anomalies between the trends and the most used languages using the ELK suite [18]. The research concluded with an architecture that integrates data from Big Query and is analyzed with Elasticsearch, Logstash, and Kibana, which determine the correlation between trends and anomalies in developing technologies. Despite this, the research did not delve into the integration of Big Query with Elasticsearch, mentioning that only the API for Python of the Google Platform is used, without saying the use of Cloud Functions.

Besides, the article "Investigation of Architecture and Technology Stack for e-Archive System" [19] investigated the technologies and architecture models necessary for an electronic file system. This induces the OASIS architecture model for the system development and details the different types of architecture possible to use as Web-Queue-Worker, Microservices, and others. It also defined the communication processes and the potential technologies for the analytics part. MongoDB and Azure were mentioned. Elasticsearch is a distributed search engine necessary for quick query and search in the model. The article concluded with a comparison of different styles of architecture and integration that can be used to develop the electronic file system. However, it did not show a conclusive implementation for the system, nor did it delve into the Cloud architecture required for data ingestion by search engines such as Elasticsearch.

On the other hand, the research "Integrated Analytics for IIoT Predictive Maintenance using IoT Big Data Cloud Systems" [20] described the design and addition of Big Data Systems on cloud solutions for IoT systems. For this, they take advantage of serverless functions, cloud services, and domain knowledge to support dynamic interactions between human resources and equipment maintenance software. Among the technologies used are: Google Cloud Functions, Apache Nifi, and Hadoop Spark. A system was obtained, whose architecture starts from the BTS (Base Transceiver Station), which, through sensors and IoT gateways, sends the data to storage sources such as PostgreSQL and OracleDB. These are then sent through Google Cloud Functions to components intended for analytics such as Apache Nifi, Google Big Query, or Hadoop FS. The article also presented serverless prototypes using AMQP, Apache Spark, or Google Pub/Sub to integrate different industrial IoT devices throughout the company.

In contrast to our research, the article proposed a high-cost architecture, which manages other data storage instances and different architectures to maintain communication between the storage and analysis systems. Therefore, the solution has high complexity. Likewise, it does not integrate Elasticsearch into the final proposal of its analytics system.

To summarize, the studies presented cover Big Data integration architectures to be consumed by systems of the ELK suite and focus on solutions that use cloud services to guarantee scalability to a greater extent. Some used Elasticsearch as a means of distributed search due to its great potential when using the inverted index to index documents. In contrast, others proposed using Pub/Sub architectures to communicate the proposed systems' different services.

The present investigation differs from the previous ones by integrating the previous concepts through the Pub/Sub architecture. In that way, scalability, reliability, and availability are gained. Also, by optimizing the integration of Google Big Query with Elasticsearch through Cloud Functions, we achieve a stable solution that is easy to implement in different areas where it is needed.

III. CLOUD ARCHITECTURES AND NON-RELATIONAL DATABASES

A. PUB/SUB ARCHITECTURE

Software architecture is defined as a strategic design [21] that supports related activities to meet the business objectives aimed at by the software to be developed. It is intrinsically related to global requirements, and its solution is implemented on programming paradigms, architectural styles, standards based on aspects of software engineering, cybersecurity, and legal regulations [21].

Global Software Development (GSD) is a trend lately, representing the development of applications through a globally distanced team [22]. This implies that the integration of the same has to be managed to reduce communication problems and improve control and the use and design of an architecture that simplifies the coordination between distributed teams.

A Pub/Sub system works as a type of middleware with a client/multi-server architecture, which provides personalized and disseminated information by identifying and delivering a particular event for the interested user [23]. The main components of the Pub/Sub architecture are publishers, subscribers, and brokers. The subscribers are applications that generate a subscription to indicate their interest in specific content that a publisher will cause. The publishers are the information producers who publish data in the system. Then the brokers (servers) will be responsible for relating the subscriptions with the data that they are looking for and that have been previously published [23].

There are many Pub/Sub implementations in cloud environments, and some examples of these are Google Cloud Pub/Sub [24] or Amazon Pub-Sub Messaging [25]. Among all the providers that implement this architecture, key concepts are maintained. The Pub/Sub paradigm is mainly based on topics and messages [23].

Topics are specific classes of objects for customers who are interested in subscribing. Messages are objects that are sent when data is required. This works in the following way: when a message is published on a topic, each customer with a subscription to it receives the message automatically. The messages are serialized into an array of bytes since Pub/Sub is

conceived to work in distributed systems and send the data to a distributed network set.

The data distributed through Pub/Sub can be topic-based. The messages are published on different topics. The publisher is responsible for defining the class of messages that subscribers must access and content-based. The messages are only sent to the subscriber if the content or attributes of the messages match the restrictions defined by the subscriber [23].

Google Cloud Pub/Sub defines the following types of subscription: Pull and Push [24]. Each is used for specific situations. The subscriber starts pulling subscriptions with a request to the Pub/Sub server to receive messages. Then the server responds with the message or an error if the queue is empty. Finally, the subscriber must explicitly indicate to the server that it received the messages correctly using an acknowledgement ID [24]. These subscriptions are advantageous when handling a large volume of messages and prioritizing efficiency and message processing. It is also helpful to guarantee HTTPS requests when there is no public endpoint with an SSL certificate.

In a push-type subscription, the Pub/Sub server starts the process by sending the data to the subscriber through an HTTPS request to a preconfigured endpoint [24]. The subscriber must also confirm the correct reception of the messages. On the other hand, push subscriptions are helpful when the same endpoint must process multiple topics and when it is sought that the same Pub/Sub server implements the message control flow [24].

B. CLOUD FUNCTIONS

Serverless is a recent trend, referring to web applications that react to events [26]. This type of architecture manages to execute complex and distributed applications built from simple functions without requiring the developer to manage the servers or any complex operational aspect [27]. The characteristics of a Serverless architecture can be summarized in 3 elements. Granular Billing, the service is charged per use in an execution. No operational logic, the development team does not worry about architectural issues such as the auto-scaling service, which the service provider itself already does. And Event-Driven applications are deployed that only respond to events when needed, allowing interaction with serverless applications that are short-lived [27].

A Cloud Function is a software script deployed in the providers' cloud infrastructure to execute an operation in response to an external event [26]. These are short, stateless, and only run-on demand, with a single functional responsibility. The function implements specific business logic to achieve the goal of the application [27]. Its characteristics include short life, a small input is taken, and output is typically generated after a short time). Cloud Functions are devoid of operational logic, which means that the entire operative issue is delegated to the platform. They also are context agnostic, so the function does not need to know the environment or the reason for its execution [27].

The Cloud Functions can be activated by: an event generated by the cloud infrastructure (changes in the database, file upload, the loading of an object, a new item, notifications to be sent, etc.) and a direct call from the application via HTTP or the cloud service API.

C. NON-RELATIONAL DATABASES

1) Big Query

Google Big Query is a highly scalable Data Warehouse under the Serverless architecture, which comes with its built-in query engine. This query engine is capable of executing SQL on TB of data in seconds. All performance is achieved without having to manage the infrastructure capable of lifting the service [28].

Big Query's serverless architecture allows different company parts to store their datasets and easily share them for seamless cross-departmental querying. It also allows third parties to access the data to carry out their operations if they have the permissions. Big Query can also be used for basic data warehouse workflows such as ETL (Extract, Transform, and Load). However, it accepts variations of it such as EL (Extract the data, then store it in Big Query) or ELT (Extract data, store it in Big Query, and then transforms it using Big Query's built-in query engine) [28].

Also, the platform is helpful to handle large volumes of Analytics. Big Query can store large amounts of data of different types: numeric, textual, even geospatial data. It can be done directly via the REST API to facilitate data entry or exit. Among its other applications, we have the Administration Facility. Part of the design behind Big Query is to get users to focus on application development and the results they expect from Big Query rather than infrastructure management [28].

2) Elasticsearch

Elasticsearch is a distributed real-time analytics engine, first released in 2010 [29] and designed to organize data to make it easily accessible [30]. It is developed as open-source on Apache Lucene [31], in Java [29] and is part of the ELK suite [18], where we can find other tools for analytics such as Logstash and Kibana [29]. Elasticsearch works like a distributed document warehouse, saving them in JSON format [30].

Elasticsearch's architecture is composed of [32]: document (basic Elasticsearch storage instance), index (logical storage location for documents. It can be divided into one or more Shards and is structured based on the inverted index model), Node (single running Elasticsearch instance), Cluster (group of cooperating running nodes), Shards (an index can be divided into smaller parts to increase efficiency by enabling parallelism. They can be stored on different servers), and Replicas (copies of shards, used for redundancy).

Indexes in the engine are considered databases in a relational database system. Indexes are defined as a collection of JSON documents, just as databases are a collection of tables [30]. They manage fault tolerance by redundantly copying data and maintaining high data availability [30].

The main function of Elasticsearch is search [18]. The general process begins with indexing a document when it is saved to the system. Elasticsearch keeps two fields by default: the document's original content. The other is the inverted index, which is generated by a series of processes such as word segmentation and filtering during indexing [18]. It should be noted that documents in Elasticsearch are indexed by default and do not have a schema [33], so it's not necessary defining fields for data types before adding data [30].

Now, for searching, the user enters a keyword of the document he intends to access, so Elasticsearch executes a search in the inverted index table, after which the document corresponding to the keyword is related. The result is returned to the user [18].

To perform searches in Elasticsearch, Query DSL is provided, which is the library offered by the distributed engine to make the Apache Lucene query syntax more accessible to users [34]. The advantage of using this syntax is seen when making complex queries that can be done in JSON format [35]. Similarly, it is possible to use filters such as numeric_range (numeric range), and (y), or (o), among others. All

Elasticsearch operations are performed via HTTP REST API [30], for which it provides a CRUD (Create, Read, Update and Delete operations) [29].

IV. THE ARCHITECTURE FOR INTEGRATING GOOGLE BIG QUERY WITH ELASTICSEARCH USING CLOUD FUNCTIONS

Fig. 1 shows the proposed architecture that integrates Big Query and Elasticsearch non-relational databases using the cloud components provided by Google Cloud Platforms such as Cloud Functions, Cloud Scheduler, and Cloud Pub/Sub.

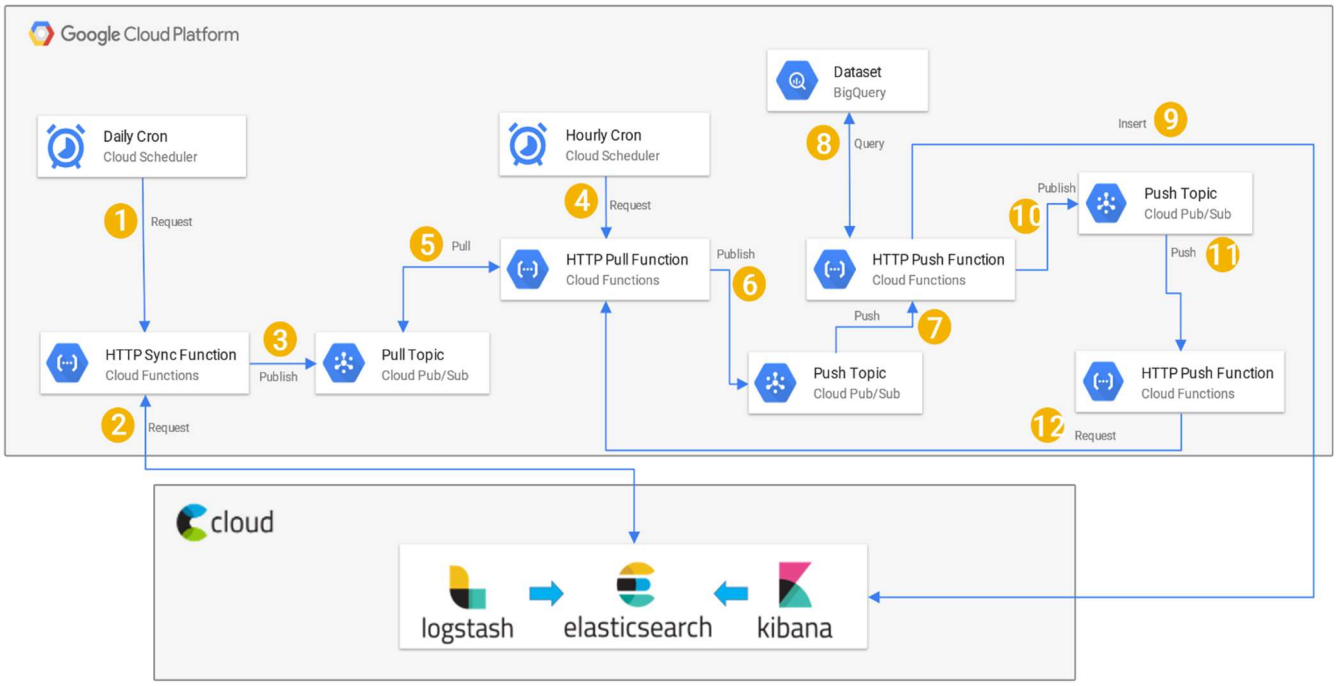

Figure 1. Architecture diagram to integrate Big Query with Elasticsearch using Cloud Functions.

The first component of the architecture is a cron that will run once a day. It will be implemented in the Google Cloud Scheduler tool executing an HTTP request to Cloud Function Sync every day at 00:00 hours. This function will be activated through an HTTP request, and its task is to request the Elasticsearch cluster to obtain the last date/ hour in which data was inserted. If it does not find dates inserted in Elasticsearch, for example, in the first execution of the function, it will have to refer to the first date found in the Big Query dataset. After obtaining the specified date (let it be the last one entered in Elasticsearch or the first of the entire Big Query dataset), the function will publish a message per day in the time interval between the date/hour obtained from Elasticsearch and the current date/hour. This topic will have a pull subscription.

The next component of the proposed architecture will be another cron that will run every hour. It will be implemented in the Google Cloud Scheduler tool executing an HTTP request to the Cloud Function Pull every hour. This function will access the pull subscription from the previous topic and extract a predetermined number of messages. These messages contain the dates to be processed for the data transfer from Big Query to Elasticsearch. If there are messages in this topic, the function publishes them in another topic with a push-type subscription. If there are no messages to extract from the topic with the pull-type subscription, the function will end.

If there are messages to process and, therefore, they have been published in the topic with the push type subscription, it will execute the main Cloud Function for each message that reaches that topic. This function will query to Big Query with the date/time it received in the message. This query will extract from Big Query all the rows that match this date/time. Then that function will gather all the rows it got from Big Query and perform a bulk insert operation to the Elasticsearch API.

The main Cloud Function will finish its execution by publishing a message on a topic. In the message published on this topic, there is a predetermined number of messages to be processed. The topic has a push-type subscription that will automatically execute a Cloud Function, passing the received message. Said Cloud Function would request the Cloud Function Pull analyzed above, which is in charge of taking a pre-established number of messages from the topic with the pull subscription and publishing them on the topic with the push subscription. With this last action, a new cycle of extracting messages with date/time from a topic begins, making a query to Big Query to obtain data according to the date/time and inserting it into Elasticsearch. The process ends when there are no more messages to extract from the topic with the pull subscription.

V. RESULTS AND DISCUSSION

A. IMPLEMENTATION, CONFIGURATION AND DEPLOYMENT

Fig. 2 shows the followed process to deploy the architecture presented in the previous section. Each of the following items will be explained below.

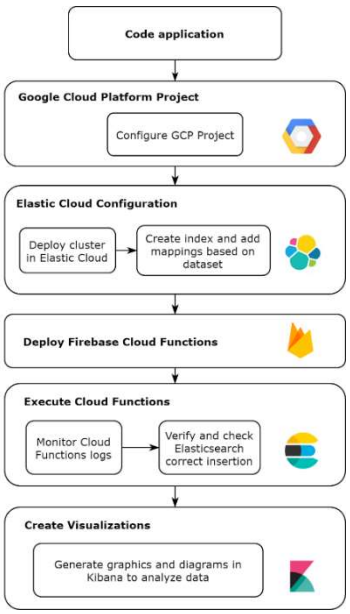

Figure 2. Organization of work carried out.

1) Code Application

This part indicates the implementation of the proposed architecture, which was carried out in Typescript to take advantage of this programming language features such as ease of debugging, faster development, and modularity when dealing with applications deployed on the web.

Client libraries of Google Big Query were used to implement search functions and query execution on datasets; Google Pub / Sub, to push and pull messages on different topics; and finally, Elasticsearch performed bulk operations and basic queries on the date.

The *iowa_liquor_sales* dataset from the public project *bigquery-public-data* was selected to perform the functional test of the implemented architecture, which has a table called sales with 1,9118,960 records to date (Aug. 15, 2021). This dataset contains wholesale liquor purchases in Iowa by retailers for sale to individuals since Jan. 1, 2012. There are wholesale liquor orders from all supermarkets, liquor stores, convenience stores, etc., with details about the store, the exact location, brand, and size of the liquor, and the number of bottles ordered.

2) Google Cloud Platform Project

It covers the configuration of the Project in Google Cloud Platform, for which after creating a project, the Pub/Sub API was enabled, and one topic of pull-type was created along with its subscription and 2 of the push type. These last two topics receive a subscription generated by the cloud functions that will be deployed later. The Google Cloud Scheduler cron needs the HTTP endpoint that refers to the function they will execute, created after functions deployment.

3) Elastic Cloud configuration

The next phase focuses on deploying a cluster in Elastic Cloud, which was created under the services of Amazon Web Services, with an I/O Optimized hardware profile in version 7.14.0. Elasticsearch has two instances, each with 120GB of storage, 4GB of RAM, and 2.2 vCPUs.

Once the Cluster was created and configured, the index was created to store the documents inserted after being extracted from Big Query. The console provided by Elastic Cloud: Dev Tools was used. Query DSL was used to perform the operations. Fig. 3 shows the operation for creating mappings based on the data types that the dataset had configured.

4) Deploy Firebase Cloud Functions

The next part contemplates the deployment of the application already with the access to the project configured in Google Cloud Platform, the dataset selected in Big Query, and the Cluster deployed in Elastic Cloud, such as the endpoints and index with which it will work in Elasticsearch. We use Firebase services since it allows the deployment of cloud functions in the Typescript language. The configuration of the Firebase CLI is done by linking our already created project in Google Cloud Platform and enabling the Cloud Functions API. To display the functions, use the command:

```
firebase deploy -only functions
```

```
PUT /sales
{
  "mappings": {
    "properties": {
      "invoice_and_item_number": { "type": "text"  },
      "date": { "type": "date", "format": "yyyy-MM-dd" },
      "store_number": { "type": "text"  },
      "store_name": { "type": "text"  },
      "address": { "type": "text"  },
      "city": { "type": "text"  },
      "zip_code": { "type": "text"  },
      "store_location": { "type": "geo_point"  },
      "county_number": { "type": "text"  },
      "county": { "type": "text"  },
      "category":{ "type": "text"  },
      "category_name": { "type": "text"  },
      "vendor_number": { "type": "text"  },
      "vendor_name": { "type": "text"  },
      "item_number": { "type": "text"  },
      "item_description": { "type": "text"  },
      "pack": { "type": "integer"  },
      "bottle_volume_ml": { "type": "integer"  },
      "state_bottle_cost": { "type": "float"  },
      "state_bottle_retail": { "type": "float"  },
      "bottles_sold": { "type": "integer"  },
      "sale_dollars": { "type": "float"  },
      "volume_sold_liters": { "type": "float"  },
      "volume_sold_gallons": { "type": "float"  }
    }
  }
}
```

Figure 3. Elasticsearch mappings creation.

This displays the functions created in our application, generating their respective endpoints. According to the architecture, two of these have to push topics as triggers, which will be invoked as soon as an *onPublish()* event, thus generating the missing subscriptions mentioned above.

5) Execute Cloud Functions

Subsequently, we execute the functions. Although we have the crons generated with Google Cloud Scheduler that can already point to the endpoints of the deployed functions, they will be performed directly to determine the functionality of the architecture. There are two endpoints to refer to: /sync and /pull.

The first function extracts the last date inserted in Elasticsearch, and when it does not find any data because it will be the first insert, it will search for reference in Big Query, obtaining the first date of the dataset. Then, it generates several messages containing one day each, taking a start date and an end date with respect. These messages are deposited in the pull topic, ending the execution of the function. After this, the Pull function is executed, extracting messages inserted in the pull topic through the subscription generated and going through all the processes designated in the architecture (Fig. 1) to insert the records Elasticsearch.

This process is monitored by the logs that the functions generate and are visible from the Google Cloud console or the Firebase console.

After finishing the Pull function execution, it was verified that the data was inserted correctly into Elasticsearch.

Figure 4. Cloud Functions logs monitoring from Firebase console.

6) Create Visualizations

Finally, with the migrated data from Big Query to Elasticsearch, visualizations were designed in Kibana (part of the Elastic suite) and added to a dashboard, through which metrics can be obtained. Data analysis can be obtained performed on the dataset *iowa_liquor_sales*.

B. RESULTS ANALYSIS AND DISCUSSION

Execution of the Sync function took 5,882 seconds, inserting 3510 messages in the pull topic. It is referred to the 3510 days between the first date of the dataset 01-03-2012 and the execution date 08-13-2021.

The 3510 messages published were synchronized with Elasticsearch using the Pull function, executed on 08-13-2021 at 13:45, ending precisely at 15:02:49. Fig. 5 represents the number of unacknowledged messages pulled in the time since the start of the Pull function.

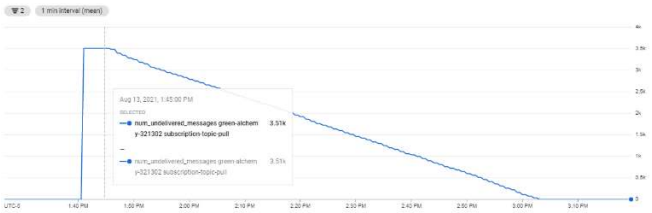

Figure 5. Unacknowledged messages pulled from pull topic.

Pull execution triggers the parallel execution of multiple threads with *OnPublishMain,* which is in charge of inserting the records extracted from Big Query in a specific date range into Elasticsearch. In Fig. 6, it is possible to see its execution, in which none of its invocations exceeded 1.5s.

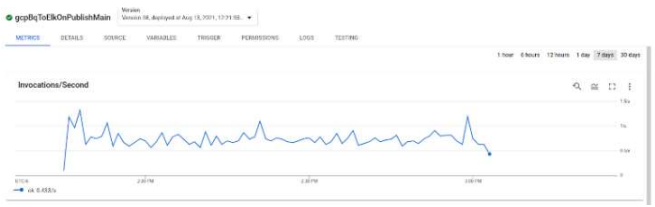

Figure 6. *OnPublishMain* function execution.

In turn, the *OnPublishSecundary* function is responsible for extracting more messages that are still in the topic pull, restarting the process completely, ending when there are no more messages in the topic. Fig. 7 shows its execution, and none of its invocations exceeded 1.5s either.

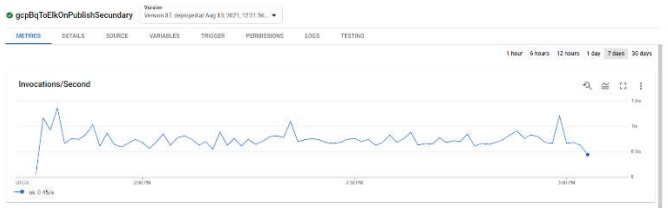

Figure 7. *OnPublishSecundary* function execution.

After complete execution, 19, 118, 960 documents were inserted in Elasticsearch.

Table 1. Obtained Measurements after insertion in Elasticsearch

Metric	Value
Documents inserted	19,118,960
(%) Effectiveness insertion	100%
Execution time	77 m.
Insertion rate docs/min	248,298.181
Insertion rate docs/sec	4,138.303

Fig. 8 shows the creation of the dashboard in Kibana, part of the ELK suite. This dashboard works with the 19,118,960 documents inserted in the index sales at the end of the Cloud Functions execution.

We added five visualizations that contain the location of the stores/buyers within the history of 2012 to date, the annual cost in taxes vs the total sales value, the percentage of the best-selling bottle packs, the number of bottles sold per month, and the average value in total sales per month since 2012.

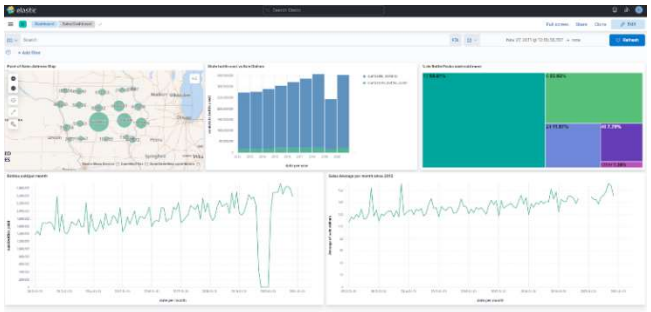

Figure 8. Kibana Dashboard.

VI. CONCLUSIONS

The extraction and analysis of large volumes of data generated over time is a latent need. Such activity has value in decision-making for companies, historical variables, and parameters for research, among other applications, so many elements must be considered to perform these operations in the best way. Although storing large amounts of data is not a problem, models must be designed based on appropriate tools to access them correctly.

The purpose of this paper is to present a Cloud Services architecture that integrates Big Query with Elasticsearch through the use of Pub/Sub and Cloud Functions. Combining these platforms, it seeks to take advantage of the capacity and ease of inserting documents in Elasticsearch, its search speed, and the comprehensive catalogue of options to create visualizations in different Dashboards that its suite offers us. In

this way, a robust and scalable application is generated to handle large volumes of data stored in Big Query and transport it to our analysis and visualization engine in Elasticsearch.

Cloud architecture was designed that uses Google Cloud Platform services, making use of Pub/Sub for the creation of topics and subscriptions which store and deliver messages to the Cloud Functions that invoke them. These Cloud Functions are in charge of performing insert operations in Elasticsearch, query data in Big Query, and handling errors in architecture. In this sense, the execution of the functions can be automated with the creation of Cloud Schedulers, generating a robust, automatic execution and completely monitorable application. This architecture facilitates error management by implementing load management and message forwarding mechanisms by itself and being self-scalable by using Google cloud services such as Pub/Sub

An experiment was performed by integrating a Big Query public dataset containing 19,118,960 records with an Elastic cloud cluster. Elasticsearch inserted as documents 100% of records received with an insertion rate of 4,138,303 documents per second, indicating the robustness and versatility of the deployed architecture. Likewise, visualizations were designed for data analysis with Kibana, part of the Elastic suite.

To summarize, large volumes of data contain valuable information in different areas, so their analysis must be carried out using carefully selected tools and methodologies. For this, the integration of storage and analysis platforms can be done by using architectures that take advantage of the potential of cloud services and carry out insertion and query operations, among others, efficiently and quickly. The architecture presented in the article achieves the integration of Big Query with Elasticsearch through Pub/Sub and Cloud Functions thanks to the Google Cloud Platform services, demonstrating its scalability, robustness, and high performance.

References

[1] Q. Na, J. Lou, Y. Yang, D. Su, J. Wu, and J. Zeng, "A big data technology-based approach to power neural network analysis," in *Proceedings of the 9th Frontier Academic Forum of Electrical Engineering*, Singapore, 2021, pp. 677–688. https://doi.org/10.1007/978-981-33-6606-0_62.

[2] Atta-ur-Rahman, S. Dash, A. Kr. Luhach, N. Chilamkurti, S. Baek, and Y. Nam, "A neuro-fuzzy approach for user behaviour classification and prediction," *Journal of Cloud Computing*, vol. 8, no. 1, p. 17, 2019. https://doi.org/10.1186/s13677-019-0144-9.

[3] Seagate Technology LLC, "Seagate advises global business leaders and entrepreneurs to sharpen focus on data critical to the success of global business impact," Business Wire, a Berkshire Hathaway company, Apr. 04, 2017. [Online]. Available at: https://www.businesswire.com/news/home/20170403006056/en/Seagate-Advises-Global-Business-Leaders-and-Entrepreneurs-to-Sharpen-Focus-on-Data-Critical-to-the-Success-of-Global-Business-Impact.

[4] N. Feng and Q. Yin, "Research on computer software engineering database programming technology based on virtualization cloud platform," *Proceedings of the 2020 IEEE 3rd International Conference of Safe Production and Informatization (IICSPI)*, 2020, pp. 696–699. https://doi.org/10.1109/IICSPI51290.2020.9332454.

[5] O. Debauche, S. A. Mahmoudi, N. D. Cock, S. Mahmoudi, P. Manneback, and F. Lebeau, "Cloud architecture for plant phenotyping research," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 17, p. e5661, 2020. https://doi.org/10.1002/cpe.5661.

[6] U. Suthakar, L. Magnoni, D. R. Smith, and A. Khan, "Optimised lambda architecture for monitoring scientific infrastructure," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1395–1408, 2021. https://doi.org/10.1109/TPDS.2017.2772241.

[7] L. dos S. Dourado, R. S. Miranda, A. P. F. de Araujo, and E. Ishikawa, "Performance evaluation of big data applications in cloud providers," *Proceedings of the 2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, 2020, pp. 1–6. https://doi.org/10.23919/CISTI49556.2020.9140855.

[8] B. Kotecha and H. Joshiyara, "Handling non-relational databases on big query with scheduling approach and performance analysis," *Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, pp. 1–5. https://doi.org/10.1109/ICCUBEA.2018.8697561.

[9] N. Newman, S. Gilman, M. Burdumy, M. Yimen, and O. Lattouf, "A novel tool for patient data management in the ICU – Ensuring timely and accurate vital data exchange among ICU team members," *International Journal of Medical Informatics*, vol. 144, p. 104291, 2020. https://doi.org/10.1016/j.ijmedinf.2020.104291.

[10] L. Chen, N. Zhang, H.-M. Sun, C.-C. Chang, S. Yu, and K.-K. R. Choo, "Secure search for encrypted personal health records from big data NoSQL databases in cloud," *Computing*, vol. 102, no. 6, pp. 1521–1545, 2020. https://doi.org/10.1007/s00607-019-00762-z.

[11] M. Bendechache, S. Svorobej, P. T. Endo, A. Mihai, and T. Lynn, "Simulating and evaluating a real-world elasticsearch system using the RECAP DES simulator," *Future Internet*, vol. 13, no. 4, Art. no. 4, 2021. https://doi.org/10.3390/fi13040083.

[12] G. Papadimitriou et al., "End-to-end online performance data capture and analysis for scientific workflows," *Future Generation Computer Systems*, vol. 117, pp. 387–400, 2021. https://doi.org/10.1016/j.future.2020.11.024.

[13] S. Ren, J.-S. Kim, W.-S. Cho, S. Soeng, S. Kong, and K.-H. Lee, "Big data platform for intelligence industrial IoT sensor monitoring system based on edge computing and AI," *Proceedings of the 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2021, pp. 480–482. https://doi.org/10.1109/ICAIIC51459.2021.9415189.

[14] G. Zhao, S. Hassan, Y. Zou, D. Truong, and T. Corbin, "Predicting performance anomalies in software systems at run-time," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 3, pp. 33:1-33:33, 2021. https://doi.org/10.1145/3440757.

[15] E. Bugingo, D. Zhang, Z. Chen, and W. Zheng, "Towards decomposition based multi-objective workflow scheduling for big data processing in clouds," *Cluster Comput*, vol. 24, no. 1, pp. 115–139, 2021. https://doi.org/10.1007/s10586-020-03208-w.

[16] J. Ding, "Development of computer-aided English listening system based on BS architecture," *Computer-Aided Design and Applications*, vol. 19, no. S1, pp. 93–104, 2022. https://doi.org/10.14733/cadaps.2022.S1.93-104.

[17] D. Gandhi, "Analyzing open source GitHub repositories towards technology acceptance model," *Pace University, the Michael L. Gargano, 18th Annual Research Day*, May 8th, 2020, pp. 1-6.

[18] X. Tian, T. Zhang, X. Zhuang, and X. He, "Research and implementation of campus network search engine based on scrapy framework and elasticsearch," 2020, pp. 4193–4198. https://doi.org/10.1109/CCDC49329.2020.9164582.

[19] H. Falatiuk, M. Shirokopetleva and Z. Dudar, "Investigation of architecture and technology stack for e-archive system," Proceedings of the *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)*, 2019, pp. 229-235. https://doi.org/10.1109/PICST47496.2019.9061407.

[20] H.-L. Truong, "Integrated analytics for IIoT predictive maintenance using IoT big data cloud systems," *Proceedings of the 2018 IEEE International Conference on Industrial Internet (ICII)*, 2018, pp. 109–118. https://doi.org/10.1109/ICII.2018.00020.

[21] M. Jaiswal, "Software architecture and software design," International Research Journal of Engineering and Technology (IRJET), vol. 6, issue 11, pp. 2452–2454, 2019. https://doi.org/10.2139/ssrn.3772387.

[22] O. Sievi-Korte, I. Richardson, and S. Beecham, "Software architecture design in global software development: An empirical study," Journal of Systems and Software, vol. 158, 110400, 2019. https://doi.org/10.1016/j.jss.2019.110400.

[23] S. Stoja, S. Vukmirovič, and B. Jelačić, "Publisher/subscriber implementation in cloud environment," 2013, pp. 677–682. https://doi.org/10.1109/3PGCIC.2013.116.

[24] Google Cloud, "Pub/Sub: A Google-scale messaging service," Google Cloud, 2021. [Online]. Available at: https://cloud.google.com/pubsub/architecture.

[25] Amazon Web Services, Inc., "What is Pub/Sub messaging?" Amazon Web Services, Inc., 2021. [Online]. Available at: https://aws.amazon.com/pub-sub-messaging/

[26] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google cloud functions," *Future Generation Computer*

Systems, vol. 110, pp. 502–514, 2020. https://doi.org/10.1016/j.future.2017.10.029.

[27] E. Van Eyk, A. Iosup, S. Seif, and M. Thömmes, "The spec cloud group's research vision on FAAS and serverless architectures," 2017, pp. 1–4. https://doi.org/10.1145/3154847.3154848.

[28] V. Lakshmanan and J. Tigani, *Google BigQuery: The Definitive Guide: Data Warehousing, Analytics, and Machine Learning at Scale*, O'Reilly Media, Inc., 2019.

[29] V.-A. Zamfir, M. Carabas, C. Carabas, and N. Tapus, "Systems monitoring and big data analysis using the elasticsearch system," *Proceedings of the 2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, 2019, pp. 188-193. https://doi.org/10.1109/CSCS.2019.00039.

[30] S. Gupta and R. Rani, "A comparative study of elasticsearch and CouchDB document oriented databases," *Proceedings of the 2016 International Conference on Inventive Computation Technologies (ICICT)*, 2016, vol. 1, pp. 1–4. https://doi.org/10.1109/INVENTIVE.2016.7823252.

[31] A. Yang, S. Zhu, X. Li, J. Yu, M. Wei, and C. Li, "The research of policy big data retrieval and analysis based on elastic search," Proceedings of the *International Conference on Artificial Intelligence and Big Data (ICAIBD)*, 2018, pp. 43-46. https://doi.org/10.1109/ICAIBD.2018.8396164.

[32] R. Kuc and M. Rogozinski, Elasticsearch Server, Packt Publishing Ltd, 2013.

[33] P. P. I. Langi, Widyawan, W. Najib, and T. B. Aji, "An evaluation of Twitter river and Logstash performances as elasticsearch inputs for social media analysis of Twitter," *Proceedings of the 2015 International Conference on Information & Communication Technology and Systems (ICTS)*, 2015, pp. 181-186. https://doi.org/10.1109/ICTS.2015.7379895.

[34] M. S. Divya and S. K. Goyal, "ElasticSearch: An advanced and quick search technique to handle voluminous data," *Compusoft*, vol. 2, no. 6, p. 171, 2013.

[35] B. Dixit, *Mastering Elasticsearch 5.x*, Packt Publishing Ltd, 2017.

***SERGIO LAUREANO GUTIÉRREZ**, Bs. Systems Engineering from National University of Saint Augustine of Arequipa, Peru. He is currently doing an internship in Sociedad Minera Cerro Verde as a Control System Analyst Apprentice and also works as a Web Developer for many software factories in Peru and Chile. He is part of international research being carried out at the National University of Saint Augustine, which focus is Smart Mobility. His current research interests include Cloud Computing, Big Data, Networking, Data Mining and the Internet of Things.*

***YASIEL PÉREZ VERA** is a researcher in the Department of System Engineering and Informatics of the National University of Saint Augustine, Arequipa, Peru. He is also working as a Head of the Software Engineering Program at La Salle University, Arequipa, Peru. He obtained his Project Management Master's and Bachelor's Computer Engineering Degree in the University of Informatics Science, La Havana, Cuba. His research interests include Machine Learning, Cloud Computing and Databases. He is a certified Cisco DevNet Associate.*