indium
Make Technology Work

# Cost-effective Metadata Logging for Cloud Composer Tasks

## A Whitepaper

# Introduction

Do you know what happens to our data after it has been processed by the application or system? How do we track changes and monitor its lifecycle? This is where metadata logging comes into picture.

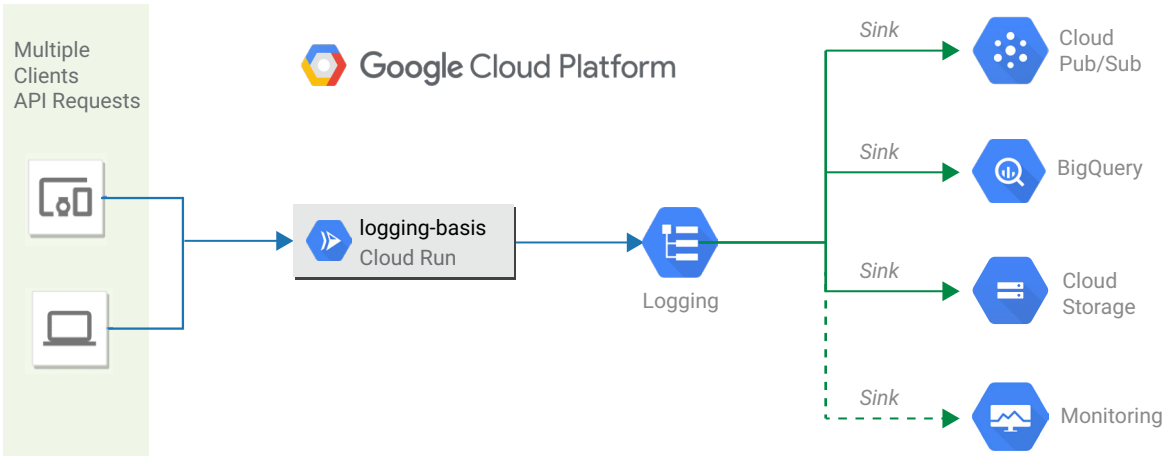
Metadata logging involves recording data about other data, such as the date and time of creation or modification, user information, and resource information. This can quickly accumulate into large volumes of data, leading to increased storage and processing costs.

As a result, organizations need to be mindful of their logging practices to ensure they are not unnecessarily generating and storing metadata.

Additionally, they may need to implement cost-saving measures such as limiting the retention period of logs or leveraging more cost-effective storage options. Failure to address these issues can result in significant financial implications for the organization.

Metadata logging allows us to track and monitor our work. Cloud Logging in Google Cloud Platform can make logging expensive if there is a lot of data moving through our pipelines. In this blog, we will explore an alternative method for metadata logging in Cloud Composer. It uses Pub/Sub, Cloud Functions and BigQuery to save money and still provide valuable log information.

Stackdriver Logging basic pipeline



# What exactly is metadata logging?

The process of recording information about our data pipeline tasks is known as metadata logging. Start and end times, input and output data, and error messages are all included. This information can be used to identify issues, improve pipeline performance, keep track of activity, and diagnose issues.

Today, a lot of businesses use cloud composer to run and schedule their pipelines.

## Here are a few explanations as to why businesses pick Cloud Composer.

- Google Cloud Platform has full management capabilities for Cloud Composer (GCP). This indicates that the Google Cloud Platform oversees the service's operation and underlying infrastructure.

  Companies are relieved of the burden of managing servers and setting up Airflow. As a result, they can focus on developing their data pipelines rather than managing infrastructure, saving time and money.

- Cloud Composer can integrate with several GCP services such as BigQuery and Cloud Storage.

  This makes it possible to create end-to-end data processing workflows. As an additional functionality, companies can also take advantage of Cloud Functions or Pub/Sub GCP services through this integration.

- Because Cloud Composer is built to scale, businesses can easily scale up and down in accordance with their needs. They don't need to manage infrastructure or allocate resources to manage large amounts of data.

- Access controls, audit logging, and data encryption are among the security features of Cloud Composer. With this feature, businesses that want to safeguard their data can rest easy.

- Utilizing Cloud Composer is usually less expensive than using our own infrastructure to run Airflow. Companies can use Cloud Composer to only pay for the services they actually use, and they can also take advantage of GCP pricing models to reduce costs.

- Cloud Composer is a fully managed service that allows companies to integrate with GCP services and take advantage of its scalability, security, and cost-effectiveness. These benefits will allow companies to build and manage data pipelines more efficiently, allowing them to concentrate on their core business.

# What is Cloud Composer?

Cloud Composer allows us to schedule, monitor, and author data pipelines. It is built on Apache Airflow and offers a powerful platform to build complex data pipelines.

Cloud Composer allows us to easily create our data pipeline workflows using directed acyclic diagrams (DAGs). We can also run them on a timer or manually trigger them.

To give our data pipes detailed log information, Cloud Composer integrates with Cloud Logging. When we run a data pipe, it automatically creates logs and sends them to Cloud Logging.

The Cloud Console or the Cloud Logging API can be used to view these logs. The cost of logging could quickly mount if we have a lot of data flowing through our pipelines.

Thus, Cloud Logging is a fully managed log service in Google Cloud Platform, that allows us to store and search for logs generated from our infrastructure and applications. This is a great tool to log events in our data pipelines.

However, it can be costly if we have multiple data pipelines continuously running and generating large volumes of logs.

Assume for the moment that the pipeline is being run with 500 GiB of logs.

## Following are the Airflow logs that Cloud Composer uses.

**Airflow logs:** These logs may be connected to specific DAG tasks.
The Cloud Storage Logs folder, which is connected to the Cloud Composer, contains the task logs. Viewing logs is another feature of the Airflow Web Interface.

For this, Cloud Storage uses two kinds of SKUs:

Standard Class for storage and Regional Standard Class A Operation to Read and Write Objects.

Standard Storage cost for 500 GiB

= 0.023 (Standard Class Cost in $) * 500 = 11.5 (in $) or, ~950 (in Rupees)

Standard Class A Ops (for ~10,00,000 Request/Day)

= 10,00,000/1000 * 0.005 (in $) * 30 (in a Month) = 5 (in $) or, ~410 (in Rupees)

**Streaming logs:** These logs are a superset from the logs in Airflow. We can access streaming logs by going to the logs tab on the Environment details page in Google Cloud Logging console or through Cloud Monitoring.

Thus, The pricing of Cloud Logging on the basis of below price provided by Google will be:

| Feature | Price |
|---|---|
| Logging ingestion | $0.50/GiB to ingest logs; this one-time fee covers up to 30 days storage |
| Logging storage | Logs that are retained for more than 30 calendar days will be charged $0.01/GiB; billing is based on retention. |

[ 500 ( GiB) * 0.50 ( in $) ] (Logging Ingestion) + [ 500 ( GiB) * 0.01 ( in $)] (Logging Storage)

= 255 (in $) or, ~ 20,500 (in Rupees)

**The overall cost can go up to 20,500 + 950 + 410 = ~21,860 Rupees/Month or, 266 (in $)**

For 500 GiB only we are getting approximately 266 $, which will increase every month, and can become a problem after some time, of course we are not going to investigate logs every day in production env.

# How should we handle the costs mentioned above?

Cloud Functions and Pub/Sub can be used to log metadata for our data pipelines. Pub/Sub allows us to send and receive messages between independent applications. It will be used to keep a list of messages that indicates when a pipeline is finished. A Cloud Function will be activated once a message has been received. This will save the result to the tables in BigQuery.

Before jumping into the solution, let's understand what exactly Pub/Sub and Cloud Functions are:

To send and receive messages between various applications, we can use Google Cloud Pub/Sub. Scaling distributed systems and microservices is made possible by this fully managed real-time messaging service.

It is very scalable and reliable to use Pub/Sub. It has many other applications and can be used to create event-driven systems and real-time messaging applications.

In order to publish messages, topics must be created in Pub/Sub. These topics have subscription options that allow users to receive messages in real time.It has the ability to send messages both within and between applications.

It can also be used to instruct Cloud Functions to react to messages that come in.

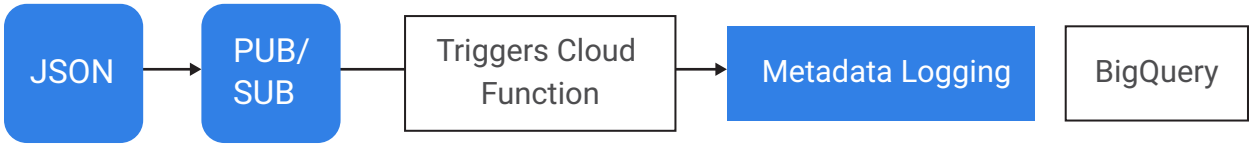
While using Google Cloud Functions, you can run code in response to events and HTTP requests without worrying about infrastructure or scaling. Pay for only the resources that are actually used by your functions.

We can write code in Python or Node.js and then deploy it to Google Cloud using Cloud Functions. Events like incoming messages, modifications to Cloud Storage objects, or HTTP requests can all start functions.

Using Cloud Functions, we can process messages that are sent to a Pub/Sub subject and decide what to do next based on the content. This could entail sending emails, updating databases, or turning on another API.

Systems that are highly scalable, trustworthy, and event-driven can be built using Cloud Functions and Google Cloud Pub/Sub. These services are fantastic for creating stable, adaptable architectures.

## Solution Architecture




## Here's how it works:

- A message is sent to Pub/Sub when a pipeline has completed its run.
- Pub/Sub adds the message into a queue.
- A Cloud Function is activated.
- The Cloud Function processes the message and extracts metadata which is saved in the tables.

The generic code to get the message as dataframe from pipeline to trigger cloud function through Pub/Subtopic:

```
def pubsub_pipeline(list_dfs, columns):
        row_counter = 0
        streaming_rows = []

    # Json Converter
        def json_converter(o):
                if isinstance(o, decimal.Decimal):
                        return o.__str__()
                if isinstance(o, datetime.datetime):
                        return o.__str__()

    # Collecting Message and sending to Queue to trigger cloud funcion
        for delta_row in list_dfs:
                streaming_row = dict(zip(columns, delta_row))
                last_row = streaming_row
                streaming_rows.append(streaming_row)
                row_counter += 1

        if row_counter > 0:
                streaming_json = json.dumps(streaming_rows[0], default=json_converter)
                future = publisher.publish(topic_path, streaming_json.encode("utf-8"))

                subscription_path = subscriber.subscription_path(project_id, subscription_id)

                def callback(message: pubsub_v1.subscriber.message.Message) -> None:
                        print(f"Received {message}.")
                        message.ack()

                streaming_pull_future = subscriber.subscribe(subscription_path, callback=callback)
                print(f"Listening for messages on {subscription_path}..\n")
                future.result()
                streaming_rows = []

                row_counter = 0
```

It will trigger cloud function which will save the message to BigQuery

So, where we go for cloud function:

```
# Set up the Pub/Sub subscriber client
subscriber = pubsub_v1.SubscriberClient()

# Set up the BigQuery client
client = bigquery.Client()

# Define the subscription and topic names
subscription_path = subscriber.subscription_path(
   '[PROJECT_ID]', '[SUBSCRIPTION_NAME]')
topic_path = subscriber.topic_path('[PROJECT_ID]', '[TOPIC_NAME]')

# Define the Cloud Function that will write the message to BigQuery
def write_to_bigquery(data, context):
        # Get the message data
        message_data = data['data']

        # Convert the message data from bytes to a string
        message_str = message_data.decode('utf-8')

        # Parse the message data as JSON
        message_json = json.loads(message_str)

         # Create a BigQuery table reference
        table_ref = client.dataset('[DATASET_NAME]').table('[TABLE_NAME]')

        row_to_insert = {'column_name': message_json['value']}

        # Insert the row into the BigQuery table
        errors = client.insert_rows(table_ref, [row_to_insert])
        if errors:
                print(errors)
        else:
                print('Data has been inserted into BigQuery successfully!')

# Set up the Pub/Sub subscriber to trigger the Cloud Function
def receive_message_from_pubsub():
          def callback(message):
                # Call the Cloud Function to write the message to BigQuery
                write_to_bigquery(message.data, None)

        # Acknowledge the message to remove it from the subscription
        message.ack()

        subscriber.subscribe(subscription_path, callback=callback)

        # Keep the program running
        while True:
                time.sleep(60)

if __name__ == '__main__':
        receive_message_from_pubsub()
```

# How is cost saving achieved?

For Pub/Sub –

Let us assume 15 GiB messages move to BigQuery, so Cost will be ~47

( in $) or ~3800 ( in Rupees/Month).

For Cloud Function costs will be:

Let 's assume Invocations as: 10,000,000

Compute Time

(128 MB / 1024 MB/GB) x 0.3s = 0.0375 GB-seconds per invocation

(200 MHz / 1000 MHz/GHz) x 0.3s = 0.0600 GHz-seconds per invocation

10,000,000 invocations x 0.0375 GB-seconds = 375,000 GB-seconds per month

10,000,000 invocations x 0.0600 GHz-seconds = 600,000 GHz-seconds per month

Networking: None

= ~ 8(in $) or, 656 (in Rupees/ Month)

**For BigQuery Storage Cost:**

500 GiB * 0.02 (In $) = 10 ( in $) or ~820(in Rupees /Month)

**So total cost will be: 3800 + 656 + 820 = 5,276 ( in Rupees /Month) or, ~ 65 (in $/Month)**

500 GiB * 0.02 (In $) = 10 ( in $) or ~820(in Rupees /Month)

**So total cost will be: 3800 + 656 + 820 = 5,276 ( in Rupees /Month) or, ~ 65 (in $/Month)**

# Conclusion

We decided against using insert scripts in the pipelines to load the metadata log into tables because doing so might cause the composer environment to become overloaded if we have several pipelines running concurrently in the composer. Therefore, we can simply redirect the logs or messages in the queue data structure by using pub/sub, which will aid us in overcoming the increased loads.

As a result, Cloud Functions and Pub/Sub can be used to log metadata for Cloud Composer data pipelines. Without affecting the calibre of the data, it provides, disabling cloud logging can save you 70% of your money. By following the advice in the blog, one can ensure that his data pipelines function effectively and smoothly.


USA
Cupertino | Princeton
Toll-free: +1-888-207-5969

INDIA
Chennai | Bengaluru | Mumbai | Hyderabad
Toll-free: 1800-123-1191

UK
London
Ph: +44 1420 300014

SINGAPORE
Singapore
Ph: +65 6812 7888

www.indiumsoftware.com

For Sales Inquiries
sales@indiumsoftware.com

For General Inquiries
info@indiumsoftware.com