International Journal for Innovative Engineering and Management Research
PEER REVIEWED OPEN ACCESS INTERNATIONAL JOURNAL
www.ijiemr.org

IJIEMR Transactions, online available on 21th Dec 2024. Link

https://ijiemr.org/downloads.php?vol=Volume-13&issue= Issue12

DOI:10.48047/IJIEMR/V13/ISSUE12/61

Title: " EVENT-DRIVEN ARCHITECTURE IN PRACTICE: ADDRESSING PRODUCTION ISSUES IN GCP PUB/SUB DEPLOYMENTS "

Volume 13, ISSUE 12, Pages: 455 - 471

Paper Authors
Sulakshana Singh

EVENT-DRIVEN ARCHITECTURE IN PRACTICE: ADDRESSING PRODUCTION ISSUES IN GCP PUB/SUB DEPLOYMENTS

Sulakshana Singh
Sr Software Engineer

Abstract
Event-Driven Architecture has been the leading design pattern that develops scalable, responsive, and loosely coupled systems, mostly within the realm of cloud-native environments. This paper delves into the implementation aspects of EDA in using Google Cloud Platform's Pub/Sub: a fully managed messaging service that simplifies event-driven system development. Though GCP Pub/Sub presents huge advantages when building a distributed system, its actual use in a production environment gives rise to several problems. These are issues concerning message delivery reliability, latency, system scalability, fault tolerance, and efficient handling of high-throughput events. The paper explores those common problems of production and will provide actual solutions to them, including exploring the key ones such as network failures, message duplication, and the performance bottleneck. Strategies regarding maintaining availability and system reliability are examined, along with techniques about optimizing Pub/Sub configurations towards better throughput and lower latency, and eventually ensuring an efficient processing rate of the messages. It provides some best practices of how to keep event-driven systems scalable in regard to how they handle keeping data consistency intact, message loss-free, and provide proper error-handling techniques. Scaling is discussed when dealing with big numbers of messages in Pub/Sub with regard to infrastructures as well as approaches toward effectively scaling both infrastructure and applications. The present work aims to be a holistic guide for engineers and architects working on implementing event-driven systems using GCP Pub/Sub by providing actionable recommendations to deploy, operate, and optimize such systems in production.

Keywords: Event-Driven Architecture, GCP Pub/Sub, Message Delivery Reliability, Scalability, Fault Tolerance and Event Processing.

1. Introduction

Event-Driven Architecture is the design paradigm where systems are event-driven in nature. Such a structure enables systems to react instantly to whatever change, action, or signal happening within their environment. Producers in such architecture create events; these events are captured, transmitted, and processed by consumers. The whole system architecture is meant to support asynchronous communication in which consumers and producers remain decoupled so that each independent component can operate with minimal lead time for reaction to events. In modern, large-scale applications, especially those requiring real-time responses or with high availability needs, EDA [1] provides quite a few benefits: It enhances scalability, flexibility, and resilience. The model is highly relevant to cloud environments where a distributed and loosely coupled system is often needed to handle diverse loads and unexpected spikes in demand.

Google Cloud Pub/Sub is a fully managed event-driven messaging system that rather directly supports such architectures, decoupling both event producers and consumers while ensuring scalability, fault tolerance, and low-latency messaging [2].

This section introduces the concept of EDA and its relevance to modern cloud-based applications. It lays the groundwork for understanding how systems can leverage event-driven architectures to handle large-scale data and real-time analytics, particularly in the context of Google Cloud Pub/Sub, which serves as the messaging backbone for such architectures. The rest of the paper is organized as follows: Section 2 presents EDA and describes its basic principles and advantages, mainly in cloud-based systems. Section 3 addresses Google Cloud Pub/Sub in particular: architecture and features for scalable, event-driven communication. Section 4 describes some production challenges encountered in real-world Pub/Sub deployments related to message delivery reliability, latency [3], and fault tolerance and provides strategies for overcoming those problems. Section 5 discusses optimization techniques to improve Pub/Sub configurations for better scalability and performance. Network failures and message duplication are discussed in Section 6 with solutions such as message acknowledgment management and deduplication for reliable delivery. Finally, Section 7 sums up the paper by summarizing some key findings, recommendations, and potential areas of further research, highlighting advancement in automation, predictive scaling, and machine learning integration into event-driven systems. This architecture makes sure that Pub/Sub is explored wholly in production environments.

2. Overview of Google Cloud Pub/Sub for Event-Driven Systems

Google Cloud Pub/Sub is a fully managed messaging service designed for the exchange of events among distributed systems. Pub/Sub will be the communication layer to allow event producers to send messages to topics, that are then delivered in real-time to one or more subscribers. Each subscriber is able to act upon such messages in a decoupled and asynchronous manner which is essential for creating highly scalable, fault-tolerant, and responsive systems. The core architecture of Google Cloud Pub/Sub consists of three major components: topics, subscriptions, and messages. Producers publish messages to topics, and subscribers consume messages from those topics. Pub/Sub guarantees at least once delivery of messages and allows for both push and pull subscriptions to cater to different processing needs. It can handle extremely high throughput and low latency, supporting millions of messages per second [4].

Pub/Sub also integrates easily with other Google Cloud services, such as Google Cloud Functions, Cloud Dataflow, and BigQuery. With these integrations, it's a great choice for event-driven pipelines, real-time data processing systems, and complex machine learning workflows. This section discusses Google Cloud Pub/Sub in depth, showing how it plays a role in building event-driven systems. It covers the service's architecture and its main features, discussing its ability to scale so that complex, data-driven applications are easy to manage by an organization.

3. Prime Production Challenges on GCP Pub/Sub Deployments

Although GCP Pub/Sub provides significant advantages in terms of scalability and flexibility, there are several challenges in deploying it in a production environment as in figure 1. These challenges must be addressed to ensure the system works efficiently under real-world conditions, especially when handling large amounts of events in a distributed manner [5].

3.1 Message Delivery Reliability

Message delivery reliability is one of the most critical aspects of event-driven systems, especially when utilizing cloud services like Google Cloud Pub/Sub. Ideally, messages that are published to the Pub/Sub system should be delivered to all subscribers with no loss. Practically, however, this is a challenging aspect of reliability in the system because of many factors, including network failures, hardware failures, and resource constraints.

Google Cloud Pub/Sub is designed to guarantee at least once delivery of messages, meaning that each message is delivered at least one time to each subscriber. This guarantees that the message will reach the intended recipient in case of a network failure or temporary breakdown of a component. "At least once" delivery, however introduces the risk of message duplication and requires proper handling to avoid inconsistent states in the system. Network issues may make the delivery of messages delayed or, worse, cause a loss of messages. For example, if a message is sent out but is not acknowledged owing to a temporary network failure, Pub/Sub will try to resend the message. This ensures reliability but also contributes to delays in processing or, worse still, duplication of messages. To address the same issues, Pub/Sub provides such capabilities as acknowledgment deadlines and re-acknowledgments of messages. In practice, reliability is built from an efficient message acknowledgment system to provide confirmation of messages or retry them without causing undue flooding in the system through redundancy [6].

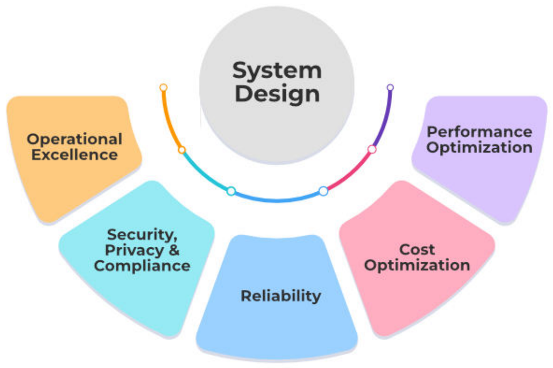

Figure 1: Challenges on GCP Pub/Sub Deployments

Another aspect of delivering the message reliably is ensuring adequate throughput of the system under delivery and processing pressure for a large number of messages, which can be sent during the increase in the number of messages. Systems implementing High throughput must be implemented to sustain bursts of data without compromising the integrity of the messages. Pub/Subscription offers horizontal scaling, meaning subordinates and backpressure mechanism for not flooding the system. It ensures that the messages are not lost nor delayed beyond [7] a certain threshold even at high loads. In addition, Pub/Sub integrates with other Google Cloud services, like Cloud Monitoring and Cloud Logging, to give visibility into message flow and delivery. These services help detect issues in message delivery early so that administrators can quickly address any network failures or delivery delays. With monitoring tools, teams are able to track metrics like acknowledgment rates and latency that are important for ensuring reliable message delivery and improving overall system health.

3.2 Latency and Throughput Control

Two critical performance metrics for event-driven architectures at deployment are latency and throughput. Latency refers to the difference in time between sending out a message and when it reaches the consumer. Throughput is the number of messages a system can process within a given time. Achieving the appropriate balance between the two elements is critical in order for the system to achieve its peak in real-time conditions, especially in high-performance environments with large amounts of data. Latency becomes highly relevant in applications reliant on the real-time process [8], such as fraud detection, monitoring systems, and dynamic content delivery. That means the latency will strongly affect the usability and productivity of the event-driven architecture. For instance, in applications of IoT-based systems where sensors send data in real time, any type of latency in the delivery of the message will result in outdated information processing or may miss the event, rendering the system unreliable.

To achieve low latency in a Pub/Sub deployment, it is necessary to optimize the message production and consumption processes. Pub/Sub provides numerous configuration options, including the acknowledgment deadline and the number of concurrent subscribers, which should be tweaked to optimize the flow of messages through the system. Regional and multi-regional subscriptions also enable faster message delivery by reducing the distance between producers and consumers. However, low latency sometimes conflicts with high throughput. In high-throughput systems, large volumes of messages are being sent and processed at the same time, which increases the time it takes to deliver individual messages. Therefore, Pub/Sub allows horizontal scaling of publishers and subscribers to manage this tradeoff. By adding more subscribers and using load balancing techniques, the system can scale up to handle higher message volumes without compromising performance. Adjusting the message batching settings within the system can improve latency and throughput. Batching allows the grouping together of several messages prior to sending them, thus reducing overhead while improving throughput. On the other hand, batching messages can result in latencies as the system has to wait until there is a number of them to be sent at a given time. Optimizing batch size and delivery intervals requires caution so that the optimum balance can be reached between throughput and latency. Pub/Sub integrates with the other GCP tools to guarantee that latency and throughput can be optimized for a given use case. For example, there are tools available like Cloud Pub/Sub Monitoring, which provides insights into performance metrics, thus enabling the identification of bottlenecks in the system between the network, the message processing pipeline, or resource contention. Organizations can continuously monitor these metrics and adjust configurations to ensure their Pub/Sub deployments maintain low latency and high throughput [9].

3.3 Fault Tolerance and Error Handling

Any distributed system needs fault tolerance and error handling, but more critically so in an event-driven architecture on cloud platforms such as Google Cloud Pub/Sub. Due to reasons like network failures, resource exhaustion, software bugs, or hardware faults, components are more likely to fail in large-scale production environments. Ensuring these failures do not cascade through the system is necessary for ensuring reliability and performance in architecture.

Pub/Sub uses several mechanisms of fault tolerance, including retry policies, dead-letter queues, and message acknowledgment mechanisms. Among its core features is the automatic retrying mechanism for messages that failed to get acknowledged within a specified time frame. In this manner, transient failures cannot cause messages to be lost. Nevertheless, retries can cause some messages to be duplicated and must therefore be handled to prevent data inconsistency. Dead-letter queues (DLQs) are another very important tool in Pub/Sub's fault-tolerant architecture. DLQs give a mechanism to capture undeliverable messages after multiple retry attempts. Those messages are isolated in a separate queue for further analysis and troubleshooting, and this way, the administrator is able to find the causes of delivery failures. DLQs are especially useful in situations where failure is not temporary; for example, when the subscriber becomes unresponsive or suffers from a persistent error [10].

Good error handling within Pub/Sub also depends upon proper configuration of acknowledgement deadlines. Upon the failure to acknowledge a message within such a deadline, Pub/Sub assumes that the message is not delivered and hence attempts to redeliver. Although this will prevent missing messages, it could potentially cause many duplicate messages, and these need to be accounted for during system design to properly manage duplicates, including techniques such as idempotent processing. Besides retries and DLQs, Pub/Sub offers a set of tools for monitoring and alerting that help detect errors as soon as possible. By integrating Pub/Sub with Cloud Monitoring, organizations can track system health and get notified about message delivery failures, high latency, or any other issues that could jeopardize the reliability of the system. Continuous monitoring ensures that an issue is detected before it could impact system performance or trigger significant downtime. The overall design of the system also affects fault tolerance in event-driven architectures. Distributed systems, by nature, have to be designed to be resilient with redundant components and failover mechanisms. In this case, distributing Pub/Sub workloads across multiple availability zones or regions allows the system to remain operational if one region goes down. Geographic redundancy ensures that the system remains available and fault-tolerant even in large-scale disruptions. Henceforth, ensuring fault tolerance and effective error handling in a GCP Pub/Sub deployment requires a combination of system design, configuration management, and robust monitoring. With Pub/Sub's retry policies, dead-letter queues, acknowledgment deadlines, and real-time monitoring tools, organizations can maintain the reliability and resilience of their event-driven systems. Proper handling of failures ensures that the system remains operational and that message integrity is preserved even during periods of disruption [11].

4. Optimizing Pub/Sub Configurations for Scalability

Optimization of Pub/Sub for large volume, event-driven, and real-time systems involves some considerations based on system scale. Proper handling of system performance is required for effective scaling of such a system. Throughput and load balancing are to be managed during the deployment of Google Cloud Pub/Sub on high-volume, real-time event-driven systems. Throughput represents the message processing rate. The load balancing ensures that all resources are used efficiently so that no single component gets overburdened. Thus, optimization of these two becomes important to make sure that the system remains scalable and performant even under high loads.

4.1 Horizontal Scaling and Load Balancing

Google Cloud Pub/Sub is designed in such a way that supports the high volume of messages, so horizontal scaling can be accommodated, meaning if load increases, then more resources can be added to the system without affecting its performance. Horizontal scaling is the feature wherein the processing load of the system is spread across several instances of the service, hence the system keeps its processing requirement balanced. Pub/Sub ensures that a specific component does not get overloaded while increasing the number of subscribers or services that are receiving messages so that messages keep getting delivered at a good pace. One of the most important ways of scaling in Pub/Sub is by partitioned topics. Partitioning a topic splits the stream of messages into smaller, more manageable parts. In other words, every partition acts like a separate unit that can handle a set of messages and further forward them to different subscribers or services. That is, several subscribers may pull messages from different partitions independently and process them in parallel, hence efficiently managing throughput and preventing one component from becoming a bottleneck.

Apart from partitioned topics, other strategies used to further optimize message distribution include load balancing strategies. For example, a subscriber can achieve message load balancing by making sure that the share of messages received by each subscriber is distributed in an evenly spread manner considering the processing capabilities and load of every subscriber. A subscriber that is overwhelmed can temporarily be sidelined or a new subscriber added for even load distribution. Pub/Sub supports creating many subscriptions for a single topic and can deliver messages to multiple subscribers concurrently. Subscriptions may also be configured with varying load balancing strategies depending on subscriber group requirements. Some subscribers may require higher priority or processing guarantees, whereas others can handle more traffic with less stringent timing constraints. Organizations can fine-tune message routing and load balancing based on varying system needs by using several subscriptions [12].

4.2 Parallel Processing Methods

Parallel processing is another optimization technique for throughput in event-driven systems. Pub/Sub lets messages be processed concurrently by a number of consumers thus enhancing the overall throughput of the system. To enable optimal parallel processing, use of several subscribers that allow the same message to come from multiple partitions at any one time is necessary. "Message acknowledgement" is also provided by Pub/Sub for each subscriber; hence, once it has received a message to process, the subscriber acknowledges having received the message, making the resources free to be used on new messages. The use of parallel processing techniques also reduces latency as subscribers are free to work in parallel with different messages. This can make sure if some subscriber is delayed on message processing, other subscribers have a chance to process some other messages without interference, hence the importance of real-time systems in terms of ability to process large volumes with minimal delay.

4.3 Configuring Topics, Subscriptions, and Message Acknowledgements

Optimizing topics, subscription configuration, and message acknowledgements ensures high performance and scalability. This is due to a highly scalable Pub/Sub mechanism when each of these things will be optimally integrated and used within the mechanism as in figure 2. Each one of those matters in ensuring the proper distribution and handling of huge volumes and even real-time data within it. Each topic is a natural channel through which messages are published, and subscriptions are used to allow consumers to pull messages from these topics. The number of topics and subscriptions should be well balanced so that the system does not overload any single topic or subscription with large volumes of data.

A closely related issue is partitioning topics. Google Cloud Pub/Sub enables topics to be split into multiple parts, hence evenly distributing the load and facilitating parallel processing of messages. The number of partitions created at the time of forming the topic needs to be decided based on the volume of messages to be received. It then leads to a high possibility of concurrent processing by high numbers of subscriptions but at a cost of increasing complexity of management for the system. In addition to partitioning, the number of subscriptions has to be scaled to match the number of consumers the system requires. That is, if the system deals with many subscribers who individually process messages, then it would be necessary to have a few subscriptions for every topic. This way, all subscribers will have equal opportunity to process messages without other subscribers being overwhelmed [13].

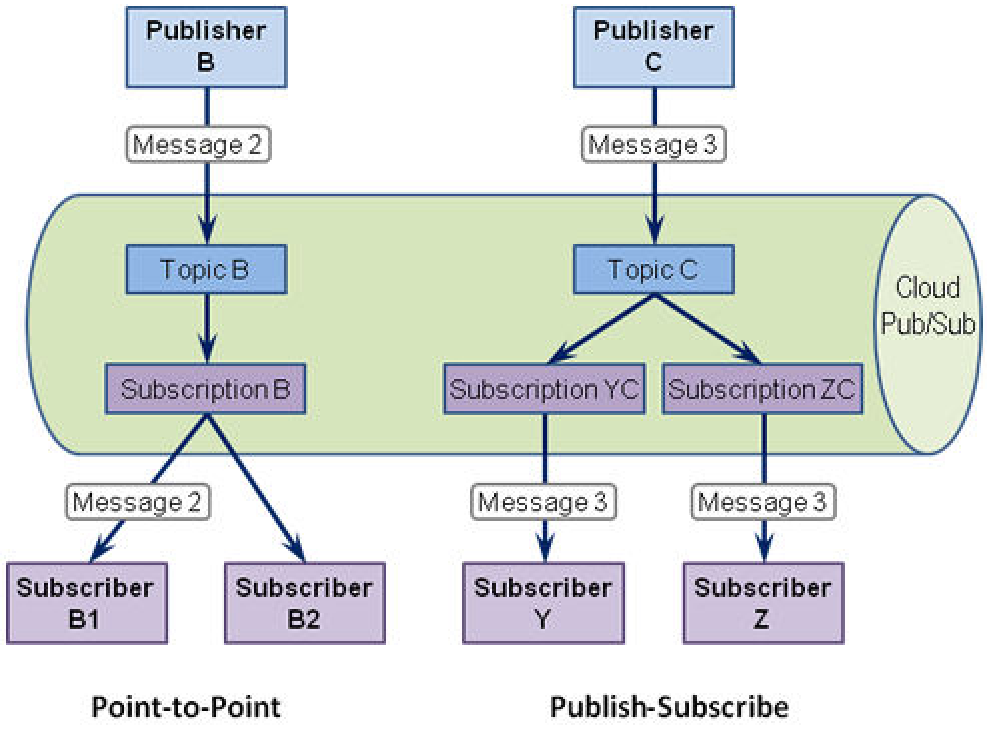

Figure 2: Configuring and auto-scaling in GC

4.4 Message Acknowledgement Configuration

It means, of course, that message acknowledgements become the basis of reliability in such event-driven systems. The Pub/Sub receives an acknowledgment for every message that's successfully handled by a subscriber. It waits for acknowledgement, failing which it'd deliver or attempt to deliver it again to another subscriber, try re-delivering it again. The acknowledgment timeout has to be adjusted along with proper deadlines for acknowledgment to optimize the performance. If the time set for acknowledgment is small then there is a likelihood of treating the message as undelivered. This would lead to retries in vain and duplication of the same message. If the acknowledgment deadline is too long, then a new message may not reach the subscribers before the deadline. So, the latency of the system can be increased.

Furthermore, developers should introduce "acknowledgement deadlines" that differ by message type. For example, for a message requiring timely processing, quick acknowledgement can be performed with lesser slack for the other low-priority messages. Fine-tuning of acknowledgement settings ensure that the system is workable without overloading any single subscriber.

4.4.1 Message Batching

The next significant configuration to consider for maximizing throughput is message batching. Batched messages can accumulate and be sent as one big batch in Pub/Sub. This way does not necessarily incur a lot of requests toward the network for a specific message delivery, thereby lowering overhead. However, this increases latency: batch sizes that are too high take longer before getting processed completely and thus delay in delivering messages to consumers. The developers should further fine-tune the batch size and intervals for delivery depending on the latency requirements and the expected volumes of messages. Lower latency comes with smaller batches but greater overhead, whereas larger batches ensure reduced overhead but might incur possible delays in the processing of messages, and hence these trade-offs must be balanced for achieving scalability while satisfying the requirements of latency as well as throughput [14].

4.4.2 Practical Techniques for Optimizing Pub/Sub Deployments

The following are pragmatics strategies for optimizing Google Cloud Pub/Sub deployments to be used in high-volume real-time data:

4.4.3 Efficient Design of Topics:

The very first thing one does is design topics to handle specific types of messages based on the characteristics of the message. This minimizes overlap in messages and allows greater scalability. For example, based on regional geographical locations, messages should be routed to separate topics to efficiently process the message in localized areas.

4.4.4 Leverage Regional and Multi-Regional Subscriptions

Pub/Sub must be configured to use regional or multi-regional subscriptions if the system has a global user base or geographically distributed services. This allows subscribers closer to the source of the message to process data faster, reducing latency and increasing throughput.

4.5 Scaling with Auto-Scaling Subscribers

Pub/Sub automatically scales to handle large message volumes. Developers can make use of auto-scaling techniques in subscribers so that the system can handle spikes in message traffic. The configuration of autoscaling for subscribers is such that changes to the number of active subscribers based on inflowing traffic is dynamic so that throughput is ensured without human intervention.

4.6 Monitoring and Continuous Optimization

The need for continuous monitoring of Pub/Sub performance with Google Cloud's monitoring tools to detect bottlenecks and inefficiencies makes it essential to track some key metrics like message delivery latency, throughput, and acknowledgment rates to fine-tune the Pub/Sub configuration. Following such strategies will help organizations maintain their Pub/Sub deployments as scalable, reliable, and performing even as the volumes of messages and the requirements to process them increase. Optimizing Pub/Sub for scalability requires careful planning and tuning of various system components, such as topics, subscriptions, and message acknowledgments. Organizations can control throughput in their event-driven systems effectively by using partitioned topics, configuring batching, changing acknowledgment deadlines, or load balancing and parallel processing techniques. Correctly configure, monitor, and constantly optimize high-performance Pub/Sub deployments that handle the complexity of high-volume real-time event processing without any decline in performance or reliability.

5. Network failures and message duplicates

Event-driven architectures are designed to handle very high volumes of events so that the system responds in real-time to its input. However, with a scaled-up deployment of the system, the issues surrounding the reliability and consistency in the delivery of events gain importance as shown in figure 3. One of the important challenges in using event-driven systems for cloud environments such as Google Cloud Pub/Sub is network failures and message duplication. However, despite the "at least once" delivery guarantee that Pub/Sub promises, duplication, loss, or delivery in an unordered way can be expected to happen during network interruptions or resource contention.

In a distributed system, components would be spread across different geographical regions or cloud environments, and network failures are always challenging. In case of such a failure, it will impact the continuous flow of messages between producers (publishers) and consumers (subscribers). Thus, these issues are critical to address for system integrity, consistency, and to avoid erroneous processing or inconsistency in state. In this section, we describe several strategies designed to mitigate the effect of network failures, message duplication, and the delivery of correct messages in event-driven systems. Strategies are mainly designed for minimization of disruptions within a system, risks of message loss, and correct message execution.

5.1 Strategies for Reliable Message Delivery

Reliable message delivery in Google Cloud Pub/Sub ensures network failures are resolved and the chances of receiving duplicate messages are minimal. Pub/Sub guarantees "at least once" delivery, which means that a message is delivered to subscribers at least once, but this can cause multiple deliveries of the same message at times. To address these problems, several strategies need to be applied to ensure the reliable delivery of messages in an event-driven architecture.

5.1.1 Message Deduplication

One of the most critical issues in event-driven systems, particularly those using the "at least once" delivery model, is message duplication. When a network failure occurs, Pub/Sub will attempt to redeliver a message. In most scenarios, this means that the subscriber will receive the same message more than once. This leads to unnecessary processing duplication, creating problems like the creation of multiple records in the database or repeating the same calculation. To avoid this, deduplication techniques must be implemented. Deduplication finds duplicate messages and does not allow these messages to be processed multiple times. Different methods can be implemented for deduplication:

- Unique Message IDs: Every published message within the system should have an ID, which allows tracing and comparison of messages through the subscribers. After the message is received, it can determine whether it already processed a message with this ID. If the message ID was already processed, then there is no need to do further processing to ensure only one instance of each message acts upon.
- Stateful Deduplication - Another technique is the keeping of a state of all last N messages that the service processed. The records could be temporarily kept either in cache or a database. On receipt of a new message, the subscriber checks his state and checks whether it had existed within the cache and/or the database. That means it won't need to process a duplicate; instead, only unique items would be processed.
- Client-Side Deduplication: In some cases, the logic to deduplicate can be left with the publisher or the application on the client-side before publishing any message to the Pub/Sub system. In many systems that have network failures coupled with high retries, they can make sure to rarely send a message more than once.

The system ensures removal of duplicate messages and the problems associated with repeated processing and data integrity. The strategies for deduplication help in filtering the duplicate messages.

5.1.2 Management of Acknowledgments

The process of acknowledgment in Google Cloud Pub/Sub plays a critical role in the reliable delivery of the messages as given in figure 3. A subscriber to the Pub/Sub automatically acknowledges a message upon the receipt of that message in the subscriber. But, if it fails to do so in the time allocated then Pub/Sub takes over the idea that the message hasn't been processed right and starts sending the same thing again. For the reliable delivery of messages, acknowledgment timeouts should be managed effectively [15].

- Optimization of Timeout: It is important to set an appropriate acknowledgment timeout depending upon the complexity of processing a message. For short processing times, a smaller timeout value might be enough to allow the system to quickly move on to the next message. In more complex operations, like database transactions or lengthy computations, longer acknowledgment deadlines are required.

- Automated retries and dead-letter queues. Pub/Sub will retry delivery when a message fails to acknowledge. In the case of a failed message that was unable to be processed following several attempts, the system should then forward the message into a DLQ where more detailed investigation can take place. It is advisable because it would avoid indefinite message retries from congesting the system. Dead-letter queues can be configured with their own retry policies and can be analyzed later to establish the issues in processing.

- Acknowledgment Tracking: To prevent premature acknowledgement of messages (when processing is not yet completed), subscribers can use acknowledgement tracking so that acknowledgement occurs only after processing is confirmed. This way, by delaying acknowledgement until after the actual processing, subscribers can ensure that a message is not marked "successfully processed" unless it really has been processed.

5.1.3 Message Ordering

For many use cases, the order in which messages are delivered to subscribers matters. Events have to be processed in an order to ensure consistency of an application, such as sequential updates on data or transactional systems. At the same time, though, Pub/Sub's "at least once" delivery guarantee can sometimes be really problematic and lead to out-of-order message delivery - most notably when retries occur because of network failures. This issue must be addressed by implementing Pub/Sub in such a manner that messages are guaranteed to be processed in correct order. Pub/Sub isn't ordered by default as it guarantees message ordering but simultaneously gives out order keys, which can deliver related messages in required sequences.

- Message Ordering Keys: An ordering key may be assigned to a message published. All messages that share the same ordering key are delivered in sequence to the subscriber. Pub/Sub may ensure that related events are processed sequentially because related events, such as updates on the same object, are minimized and therefore eliminate inconsistencies from the out-of-order message delivery.
- Global Ordering vs. Local Ordering: A choice needs to be made whether global ordering-that is, messages across all topics must be processed in sequence-or local ordering-that is, only related messages must be ordered-is necessary. This is important for Pub/Sub configuration to ensure the appropriate level of message ordering relative to application requirements and data consistency.

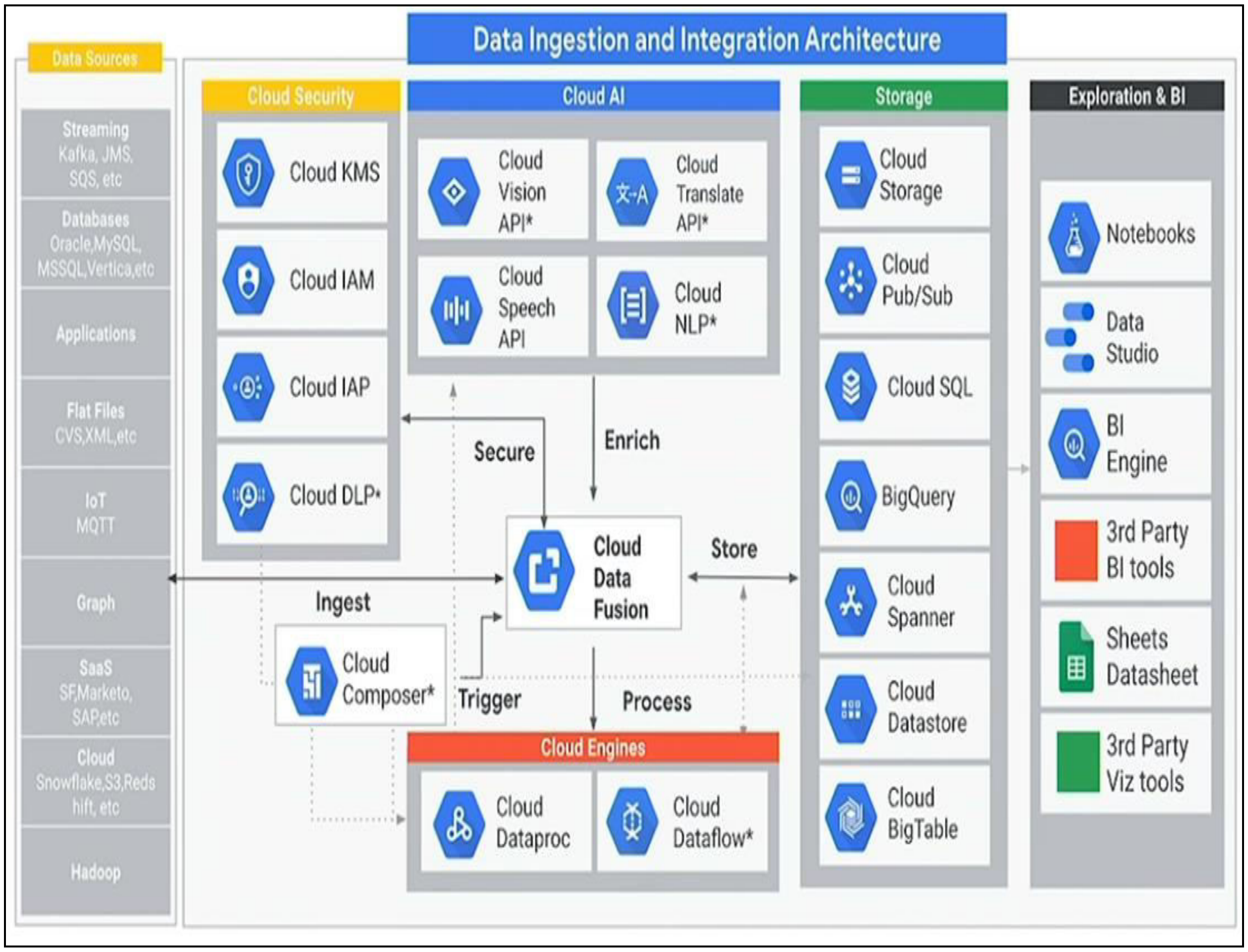

Figure 3: Simulation of data ingestion in GC without network failure

5.2 Message Duplication and Idempotence

When using an event-driven architecture with Pub/Sub, making sure that message duplication does not lead to unintended side effects is the key. That's when idempotency is needed.

5.2.1 What is Idempotency?

Idempotency refers to the ability of a system to handle repeated operations without causing unintended side effects. In the context of message processing, an idempotent system can process the same message multiple times without altering the result beyond the first successful processing. For example, if a message triggers a database update, processing the same message again should not result in duplicate database entries. Idempotent systems are especially important when working with unreliable messaging systems because they let subscribers safely process the same message multiple times without leading to an inconsistent state.

5.2.2 Idempotence in Subscribers

To say that a subscriber is idempotent, it must implement duplicate message detection such that duplicate messages are dealt with properly. Unique Message Ids: a message should be equipped with unique identifier able to be referred to and used by subscriber to understand whether the given message has previously been processed, when subscriber will receive the message, she can compare ID of given message with ID list of previous processed IDs in database or some cache, where if matched, she wouldn't bother to process to avoid its side effects.

- Database Constraints: The approach to idempotency through database constraints is such that it prevents duplication of records. For instance, if a unique constraint is put on a field, like an order ID, then the same record will not be inserted into the database multiple times, even if the same message is processed multiple times.
- Transactional Integrity: In some instances, subscribers may support message processing within a transactional context. A subscriber can utilize database transactions so that the modifications occur only if the transaction succeeds entirely. If the message is a duplicate or otherwise retried, the system is free to back out the changes without having unwanted side effects on the system.

5.3 Scalable Large-Scale Deployments with Reliable Message Delivery

Event-driven systems can handle scalability at a larger deployment by maintaining a certain level of message reliability with the challenging conditions applied during network failure, a very high-throughput workload, and a duplicated message. Message deduplication, acknowledgment management, and idempotency strategies help remove some common pitfalls. Organizations can apply the above strategies in a way to make Pub/Sub deployments very robust, reliable, and can handle the high amount of events in order not to undermine data consistency and system integrity. These best practices diminish effects of network failures and effects of message duplication and of delivering messages differently thus making a system perform and accurate for any real application.

Addressing network failures along with message duplication on an event-driven system makes keeping up the reliability and integrity of the system. With its strategies on message deduplication, acknowledgment management, and idempotent design in processing, the organization would certainly mitigate possible risks that come with the least once delivery guarantees of these large-scale and high throughput environments. This will hence advance its reliability and performance within Google Cloud Pub/Sub deployments.

6. Discussions and interpretations

Event-Driven Architecture, EDA for the past few years has represented an increasingly significant paradigm for designing scalable systems as well as flexible and loose systems. The increase of cloud-native technologies only emphasized it. Google Cloud Pub/Sub became popular in implementing EDA through offering a fully managed messaging system that scales automatically between any event producers and any consumers. However, while GCP Pub/Sub offers many advantages, the deployment of this system in real-world production environments presents several challenges that need careful attention to ensure the system operates effectively.

A key challenge in GCP Pub/Sub deployments is ensuring reliable message delivery. The "at least once" delivery guarantee can be both an advantage and a drawback. While it ensures messages are delivered, it may lead to the duplication of messages under some failure conditions. The issue of message duplication becomes all the more critical in those systems that handle sensitive data because duplicates could cause issues related to data integrity. To prevent the risks of duplicate messages being processed, organizations can take advantage of strategies such as deduplication techniques, management of acknowledgments, and implementing idempotency at the subscriber level. Another challenge in the deployment of GCP Pub/Sub is latency and throughput. Event-driven systems generally require real-time processing of messages, and any delay in message delivery or processing affects the system performance quite heavily. The throughput of the system needs to be optimized to handle high-volume events without increasing latency. Organizations can ensure the system can scale horizontally to meet the demands of large-scale systems with minimal impact on latency by partitioning topics, using multiple subscriptions, and employment of load balancing strategies.

The other important consideration is fault tolerance. Distributed systems are prone to failures due to network issues, resource exhaustion, or software bugs. To ensure that failures do not impact overall system performance requires some level of robust error handling, including retries, dead-letter queues, and auto-scaling. These help ensure high availability and allow the system to recover gracefully in case of failures.

Scalability is at the heart of the GCP Pub/Sub design, but even with such scalable infrastructure, careful tuning is required. Without appropriate tuning, systems may fail to cope well with large volumes of messages. Optimizing configurations, such as message acknowledgment times, partitioning, and horizontal scaling, allows systems to manage millions of messages with very low overhead on resources. Additionally, leveraging Pub/Sub's integration with other Google Cloud services, such as auto-scaling functions and Pub/Sub Lite for cost-effective scaling, can further improve the efficiency of the system. All these necessitate strategic planning, technical expertise, and continuous optimization. With adherence to best practices like idempotent processing, configurations optimized for scalability, as well as robust fault tolerance mechanisms, organizations may successfully deploy event-driven architectures that are both reliable and efficient.

7. Conclusion

The successful deployment of Event-Driven Architecture using Google Cloud Pub/Sub hinges on addressing several key challenges, including message delivery reliability, latency and throughput management, fault tolerance, and efficient handling of high-volume events. While Pub/Sub provides the foundation for building scalable, loosely coupled systems, its full potential can only be realized through thoughtful configuration and optimization in production environments. By applying the best practices such as message deduplication, configuring acknowledgement management, implementing fault tolerance, and ensuring efficient delivery of messages, organizations can significantly increase the reliability and efficiency of systems. In addition, configuring Pub/Sub topics and subscriptions to optimize them and scale both infrastructure and applications help ensure that the system accommodates fluctuating demands and high throughput with low latency. This paper has highlighted crucial production issues and strategies to mitigate them, providing an integral approach for engineers and architects working with GCP Pub/Sub. As organizations continue adopting cloud-native, event-driven systems, there is a growing requirement for scalable, resilient solutions that can meet the requirements of modern applications. Future research may focus on further automating these processes and improving scaling strategies as well as new emerging technologies that complement event-driven systems. With the right practices in place, GCP Pub/Sub can easily be a solid base for scalable, reliable, and high-performance event-driven architectures in the cloud.

References

1. Laigner, R., Almeida, A. C., Assunção, W. K., & Zhou, Y. (2024). An Empirical Study on Challenges of Event Management in Microservice Architectures. arXiv preprint arXiv:2408.00440.
2. Trabelsi, N., Politowski, C., & El Boussaidi, G. (2023, May). Event driven architecture: An exploratory study on the gap between academia and industry. In 2023 IEEE/ACM 5th International Workshop on Software Engineering Research and Practices for the IoT (SERP4IoT) (pp. 25-32). IEEE.
3. Al-Maamari, T. A. A. (2016). Aspects of event-driven cloud-native application development (Master's thesis).
4. Choudhary, S. K., Ranjan, P., Dahiya, S., & Singh, S. K. (2023). Detecting Malware Attacks Based on Machine Learning Techniques for Improve Cybersecurity. International Journal of Core Engineering & Management, 7(8), 88. ISSN 2348-9510.

5. Ranjan, P., Dahiya, S., Singh, S. K., & Choudhary, S. K. (2023). Enhancing Stock Price Prediction: A Comprehensive Analysis Utilizing Machine Learning and Deep Learning Approaches. International Journal of Core Engineering & Management, 7(5), 146. ISSN 2348-9510.

6. Dahiya, S., Singh, S. K., Choudhary, S. K., & Ranjan, P. (2022). Fundamentals of Digital Transformation in Financial Services: Key Drivers and Strategies. International Journal of Core Engineering & Management, 7(3), 41. ISSN 2348-9510.

7. Singh, S. K., Choudhary, S. K., Ranjan, P., & Dahiya, S. (2022). Comparative Analysis of Machine Learning Models and Data Analytics Techniques for Fraud Detection in Banking System. International Journal of Core Engineering & Management, 7(1), 64. ISSN 2348-9510.

8. Rekha, P., Saranya, T., Preethi, P., Saraswathi, L., & Shobana, G. (2017). Smart Agro Using Arduino and GSM. International Journal of Emerging Technologies in Engineering Research (IJETER) Volume, 5.

9. Suresh, K., Reddy, P. P., & Preethi, P. (2019). A novel key exchange algorithm for security in internet of things. Indones. J. Electr. Eng. Comput. Sci, 16(3), 1515-1520.

10. Bharathy, S. S. P. D., Preethi, P., Karthick, K., & Sangeetha, S. (2017). Hand Gesture Recognition for Physical Impairment Peoples. SSRG International Journal of Computer Science and Engineering (SSRG-IJCSE), 6-10.

11. Sujithra, M., Velvadivu, P., Rathika, J., Priyadharshini, R., & Preethi, P. (2022, October). A Study On Psychological Stress Of Working Women In Educational Institution Using Machine Learning. In 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT) (pp. 1-7). IEEE.

12. Laxminarayana Korada, D. M. K., Ranjidha, P., Verma, T. L., & Mahalaksmi Arumugam, D. R. O. Artificial Intelligence On The Administration Of Financial Markets.

13. Korada, L. (2024). Data Poisoning-What Is It and How It Is Being Addressed by the Leading Gen AI Providers. European Journal of Advances in Engineering and Technology, 11(5), 105-109.

14. Laxminarayana Korada, V. K. S., & Somepalli, S. Finding the Right Data Analytics Platform for Your Enterprise.

15. Anguraju, K., Kumar, N. S., Kumar, S. J., Anandhan, K., & Preethi, P. (2020). Adaptive feature selection based learning model for emotion recognition. J Critic Rev.