Using Google Cloud Pub/Sub to Integrate Components of Your Application

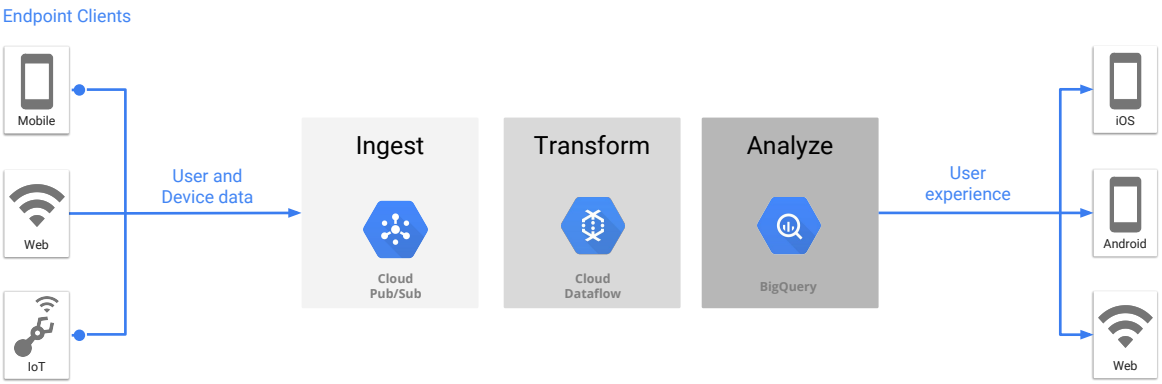
Developing Applications with Google Cloud Platform

CLOUD PUB/SUB, CLOUD DATASTORE, CLOUD BIGTABLE, CLOUD DATAFLOW, CLOUD MONITORING

DEVELOPING A BACKEND SERVICE

Version 2.0
Last modified: 2017-09-25

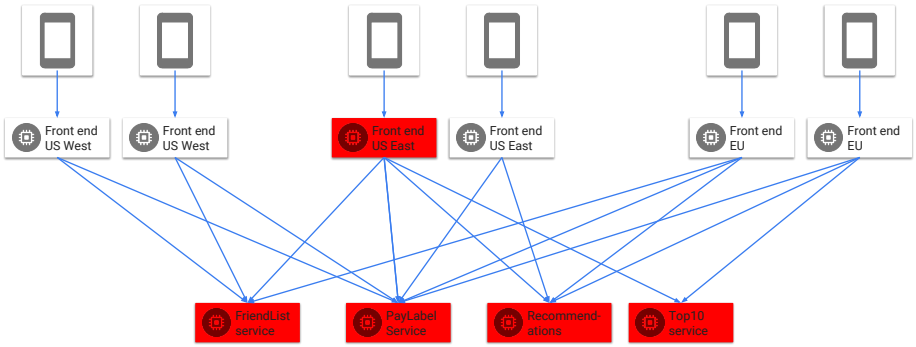
Organizations have to rapidly ingest, transform, and analyze massive amounts of data



Google Cloud

Across industry verticals, a common scenario is that organizations have to rapidly ingest, transform, and analyze massive amounts of data. For example, a gaming application might receive and process user engagement and clickstream data. In the shipping industry, IoT applications might receive large amounts of sensor data from hundreds of sensors. Data processing applications transform the ingested data and save it in an analytics database. You can then analyze the data to provide business insights and create innovative user experiences.

Organizations have to orchestrate complex business processes



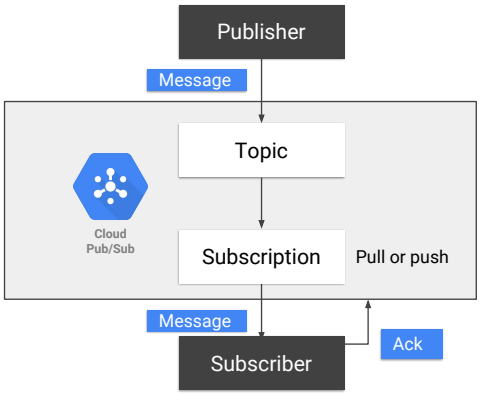
Google Cloud

Organizations often have complex business processes that require many applications to interact with each other. For example, when a user plays a song, a music streaming service must perform many operations in the background. There might be operations to pay the record company, perform live updates to the catalog, update song recommendations, handle ad interaction events, and perform analytics on user actions. Such complex application interactions are difficult to manage with brittle point-to-point application connections.

Cloud Pub/Sub enables you to scalably and reliably ingest large volumes of data. It also enables you to design loosely coupled microservices that can streamline application interactions.

Cloud Pub/Sub is a fully managed real-time messaging architecture

Cloud Pub/Sub delivers each message to every subscription at-least-once.



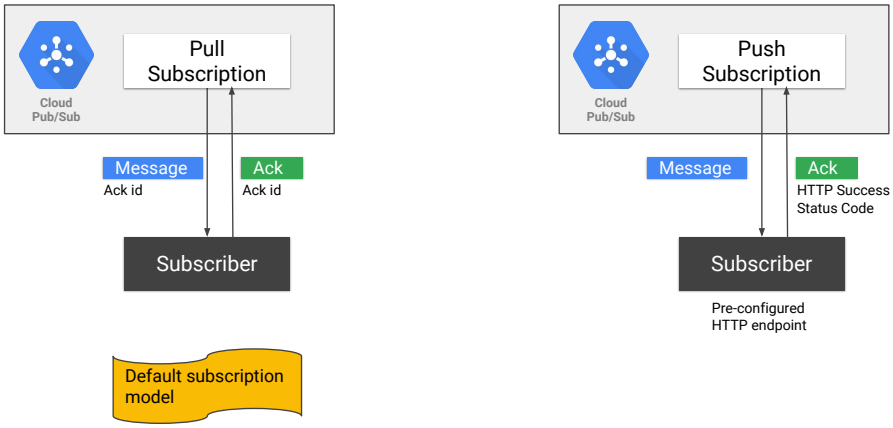
Google Cloud

Cloud Pub/Sub is a fully managed real-time messaging architecture.

An application or service that publishes messages is called a *publisher*. A publisher creates and publishes messages to a *topic*. To receive messages, a *subscriber* application creates a *subscription* to a topic. A subscription can use either the push or pull method for message delivery. The subscriber only receives messages that are published after the subscription was created. After receiving and processing each pending message, the subscriber sends an acknowledgement back to the Cloud Pub/Sub service. Cloud Pub/Sub removes acknowledged messages from the subscription's queue of messages. If the subscriber does not acknowledge a message before the acknowledgement deadline, Cloud Pub/Sub will resend the message to the subscriber. Cloud Pub/Sub delivers each message to every subscription at-least-once.

Cloud Pub/Sub enables loosely coupled integration between application components, acts as a buffer to handle spikes in data volume, and also supports other use cases.

Cloud Pub/Sub supports pull and push subscriptions



Google Cloud

Pull Subscription
In a pull subscription, the subscriber explicitly calls the pull method to request messages for delivery. Cloud Pub/Sub returns a message and an acknowledgement ID. To acknowledge receipt, the subscriber invokes the acknowledge method by using the acknowledgement ID. This is the default subscription model.

In a pull subscription model, the subscriber can be Cloud Dataflow or any application that uses Google Cloud Client Libraries to retrieve messages. The subscriber controls the rate of delivery. A subscriber can modify the acknowledgement deadline to allow more time to process messages. To process messages rapidly, multiple subscribers can pull from the same subscription. The pull subscription model enables batched delivery and acknowledgments as well as massively parallel consumption.

Use the pull subscription model when you need to process a very large volume of messages with high throughput.

Push Subscription
A push subscriber does not need to implement Google Cloud Client Libraries methods to retrieve and process messages.

In a push subscription model, Cloud Pub/Sub sends each message as an HTTP request to the subscriber at a pre-configured endpoint. The push endpoint can be a load balancer or Google App Engine Standard application. The endpoint acknowledges the message by returning an HTTP success status code. A failure

response indicates that the message should be sent again. Cloud Pub/Sub dynamically adjusts the rate of push requests based on the rate at which it receives success responses. You can configure a default acknowledgment deadline for push subscriptions. If your code does not acknowledge the message before the deadline, Cloud Pub/Sub will resend the message.

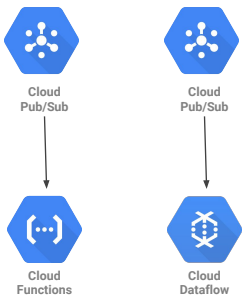
Use the push subscription model in the following scenarios:

- Environments where Google Cloud dependencies such as credentials and the client library cannot be configured.
- Multiple topics must be processed by the same webhook.
- HTTP endpoint will be invoked by Cloud Pub/Sub and other applications. In this scenario, the push subscriber should be able to process message payloads from Cloud Pub/Sub and other callers.

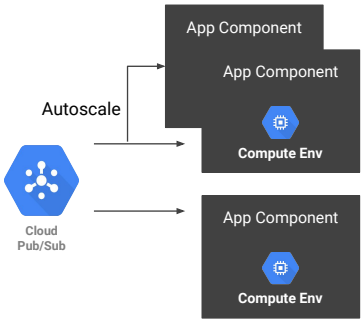
For more information about pull and push subscriptions, see https://cloud.google.com/pubsub/docs/subscriber#push_pull.

You have a choice of execution environments for subscribers

Develop highly-scalable subscribers with Google Cloud Functions or Cloud Dataflow.



Deploy subscriber on Google Compute Engine, Google Container Engine, or Google App Engine flexible environment. Autoscale based on Google Stackdriver metrics.



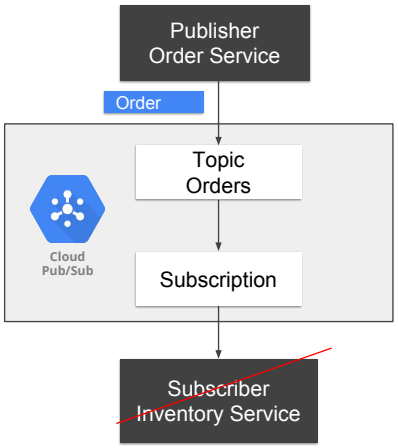
Google Cloud

You can deploy your subscriber code as a Cloud Function. Your Cloud Function will be triggered whenever a new message is received. This approach enables you to implement a serverless approach and build highly-scalable systems.

Alternatively, you can deploy your subscriber application on a compute environment such as Compute Engine, Container Engine, or App Engine flexible environment. Multiple instances of your application can spin up and process the messages in the topic and split the workload. These instances can automatically be shut down when there are few messages to process. You can enable elastic scaling using Cloud Pub/Sub metrics that are published to Stackdriver.

With either approach, you do not have to worry about developing code to manage concurrency or scaling. Your application scales automatically depending on the workload.

Use Cloud Pub/Sub for asynchronous processing and loose coupling

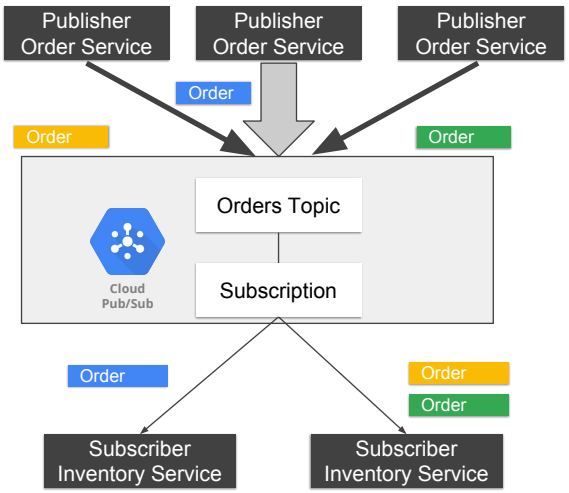


Google Cloud

Cloud Pub/Sub enables your application to perform operations asynchronously. Traditionally, one service or component of your application has to directly invoke another component to perform an operation. The call is synchronous, so the caller has to wait for the entire operation to finish. The application components are tightly coupled because both must be running at the same time.

With Cloud Pub/Sub, you can design your services to make asynchronous calls and be loosely coupled. For example, in the diagram, the Order service acts as the publisher. It publishes messages that contain order information to the Orders topic and returns immediately. The Inventory service acts as the subscriber. It might even be down when the publisher sends messages. The subscriber can start running a little later and process the messages that it has subscribed to.

Use Cloud Pub/Sub to buffer incoming data



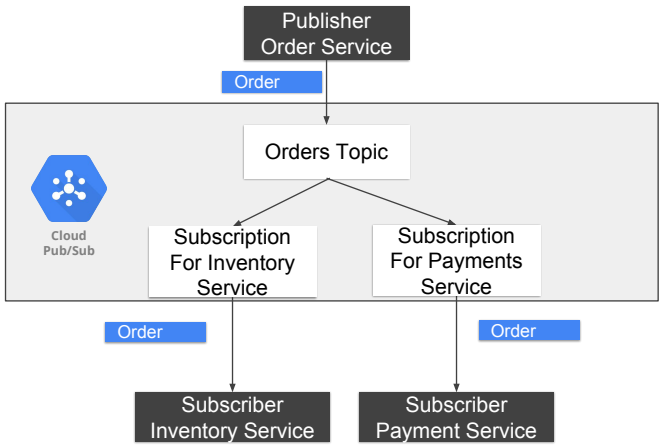
Google Cloud

Cloud Pub/Sub enables you to use a topic as a buffer to hold data that is coming in rapidly. For example, you might have multiple instances of a service generating large volumes of data. A downstream service that needs to process this data might become overwhelmed with the high velocity and volume of data.

To address this issue, instances of your data-generating service can act as publishers and publish the data to a Cloud Pub/Sub topic. Cloud Pub/Sub can reliably receive and store large amounts of rapidly incoming data. The downstream service can then act as a subscriber and consume this data at a reasonable pace. Your application can even automatically scale up the number of instances of the subscriber to handle increased volumes of data.

The diagram shows many messages with order information coming in from various sources. The Inventory service can subscribe to these messages and process them at a reasonable pace as appropriate.

Use Cloud Pub/Sub to fan out messages

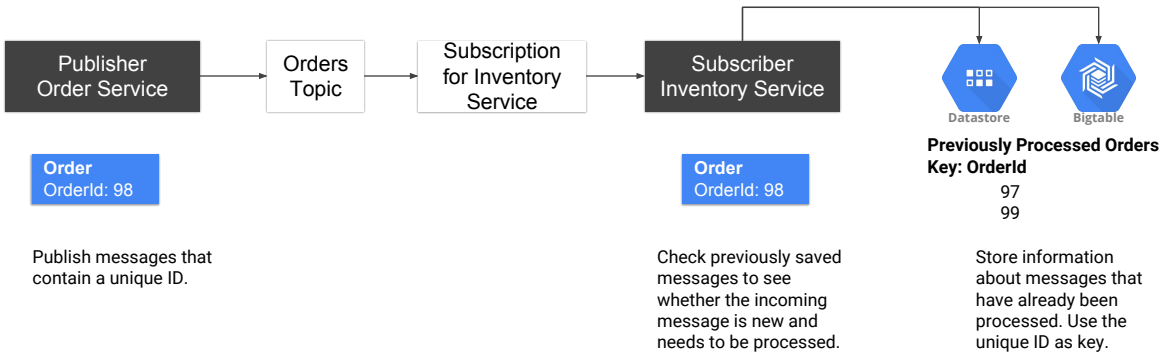


Google Cloud

With Cloud Pub/Sub, your application can fan out messages from one publisher to multiple subscribers. Instead of being aware of all services that might be interested in a particular piece of data and making point-to-point connections with all the services, the publisher can push messages to a centralized Cloud Pub/Sub topic. Services that are interested in that information can simply subscribe to the topic.
For example, in the diagram, information about orders is fanned out to the Inventory and Payment services.

Handle duplicate messages

Ensure that messages contain identifying attributes that subscribers can use to perform idempotent operations.

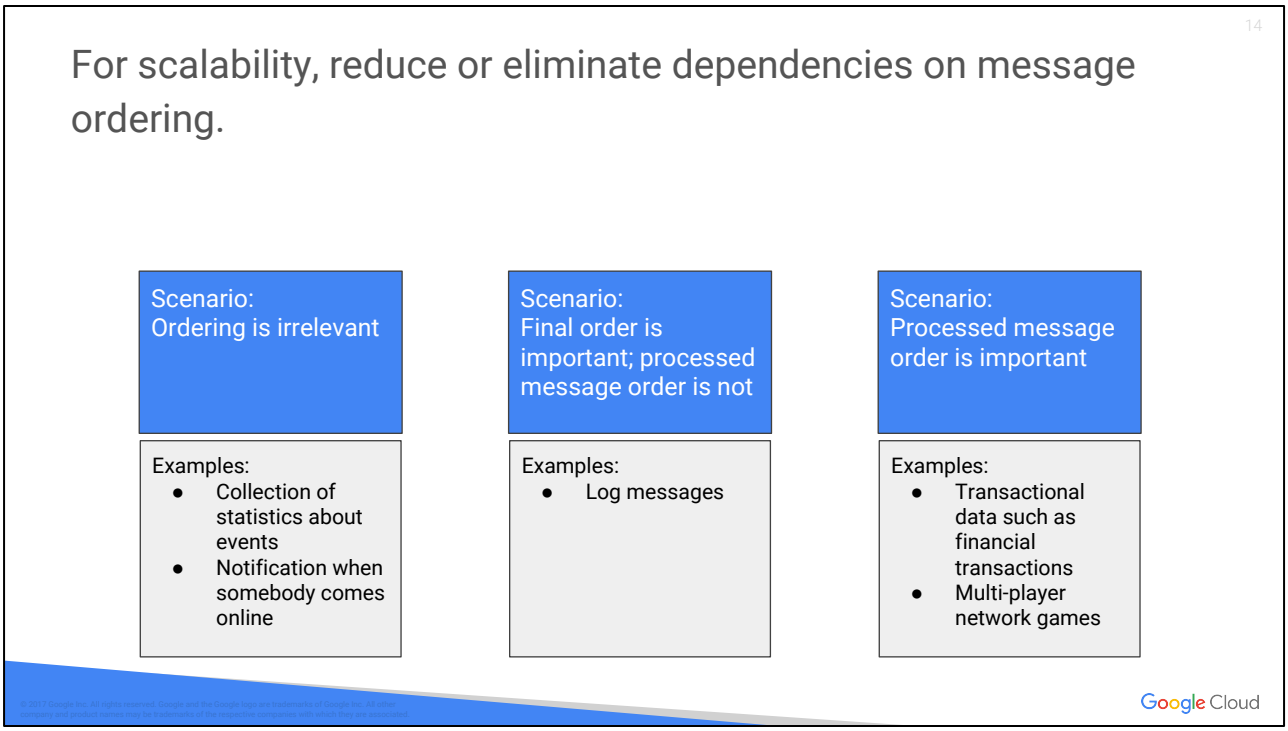


Google Cloud

Cloud Pub/Sub guarantees at-least-once delivery. This means that a subscriber can sometimes see duplicate messages. Implement your publisher and subscriber in such a way that the application can perform idempotent operations.

The publisher can publish messages with a unique id. Your subscriber can use Google Cloud Datastore or Google Cloud Bigtable to store information about messages that have already been processed. Use the unique ID as key. Whenever a message is received, the subscriber can check previously saved messages to see whether the incoming message is new and needs to be processed. If the message has already been processed, the subscriber can just discard the duplicate message.

For use cases that involve Big Data, you can use Google Cloud Dataflow PubsubIO to achieve exactly-once processing of Cloud Pub/Sub message streams. PubsubIO removes duplicate messages based on custom message identifiers or identifiers assigned by Cloud Pub/Sub. For more information about PubsubIO, see https://cloud.google.com/dataflow/model/pubsub-io.



Cloud Pub/Sub provides a highly scalable messaging architecture. The scalability comes with a tradeoff: message ordering is not guaranteed. Where possible, design your application to reduce or eliminate dependencies on message ordering.

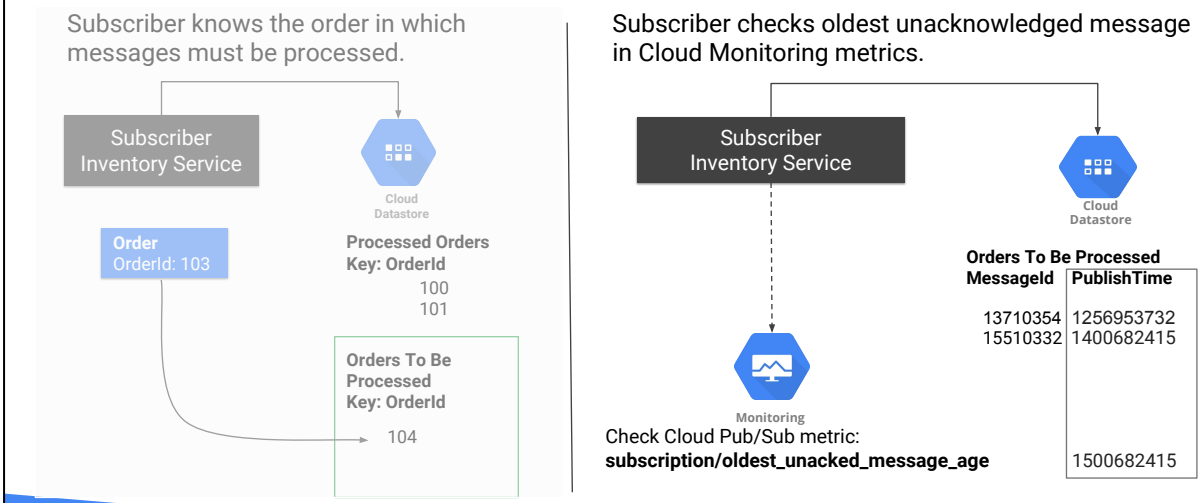
For example, message ordering is not relevant in use cases where your application is collecting statistics about events. You do not need to process the statistics-related events in order. Similarly, if five of your contacts come online around the same moment in time, you do not need to be notified about their presence in the same order that they came online.

There are scenarios in which the final order of messages is important, but the order in which individual messages are processed is not important. For example, log events with timestamps may stream into the logging service from various sources. The order of processing each log event is not important. However, eventually your system should enable you to view log events ordered by timestamp.

Cloud Pub/Sub works well naturally for use cases where order of message processing is not important.

Finally, there are scenarios where messages must absolutely be processed in the order in which they were published. For example, financial transactions must be processed in order. Similarly, in a multi-player online game, if a player jumps over a wall, lands on a monster, and recovers the lost jewel, other players must see the events in exactly the same order.

Handle message ordering for transactional data



Google Cloud

You can implement ordered processing of messages in one of the following ways:

- Subscriber knows the order in which messages must be processed: The publisher can publish messages with a unique id. Your subscriber can use Cloud Datastore to store information about messages that have been processed so far. When a new message is received, the subscriber can check whether this message should be processed immediately or the unique ID indicates that there are pending messages that should be processed first. For example, the Inventory service knows that orders should be processed in a sequence using the OrderId as key. When messages with order ids 103 and 104 are received, it checks the previously processed orders and determines that OrderId 102 is pending. It temporarily stores orders 103 and 104 and processes them all in order when order 102 is received.
- Subscriber checks oldest unacknowledged message in Cloud Monitoring metrics: The subscriber can store all messages in a persistent data store. It can check the `subscription/oldest_unacked_message_age` metric that Cloud Pub/Sub publishes to Google Stackdriver. The subscriber can compare the timestamp of the oldest unacknowledged message against the publish timestamps of the messages in the data store. All messages published before the oldest unacknowledged message are guaranteed to have been received, so those messages can be removed from the persistent data store and processed in order.

For more information about message ordering, see

https://cloud.google.com/pubsub/docs/ordering.