This document gives an overview of how subscriptions work in Pub/Sub. For details on pull and push delivery subscriptions, see the Pull Subscriber Guide (/pubsub/docs/pull) and the Push Subscriber Guide (/pubsub/docs/push).

To receive messages published to a topic, you must create a subscription to that topic. Only messages published to the topic after the subscription is created are available to subscriber applications. The subscription connects the topic to a subscriber application that receives and processes messages published to the topic. A topic can have multiple subscriptions, but a given subscription belongs to a single topic.

To learn about creating and updating subscriptions, see Managing Topics and Subscriptions (/pubsub/docs/admin).

Pub/Sub delivers each published message at least once for every subscription. There are some exceptions to this at-least-once behavior:

- By default, a message that cannot be delivered within the maximum retention time of 7 days is deleted and is no longer accessible. This typically happens when subscribers do not keep up with the flow of messages. Note that you can configure message retention duration (the range is from 10 minutes to 7 days). See Replaying & Discarding Messages (/pubsub/docs/replay-overview) for more information about the message retention setting.
- A message published before a given subscription was created will usually not be delivered for that subscription. Thus, a message published to a topic that has no subscription will not be delivered to any subscriber.

Once a message is sent to a subscriber, the subscriber should acknowledge the message. A message is considered outstanding once it has been sent out for delivery and before a subscriber acknowledges it. Pub/Sub will repeatedly attempt to deliver any message that has not been acknowledged. While a message is outstanding to a subscriber, however, Pub/Sub tries not to deliver it to any other subscriber on the same subscription. The subscriber has a configurable, limited amount of time – known as the `ackDeadline` – to acknowledge the outstanding message. Once the deadline passes, the message is no longer considered outstanding, and Pub/Sub will attempt to redeliver the message.

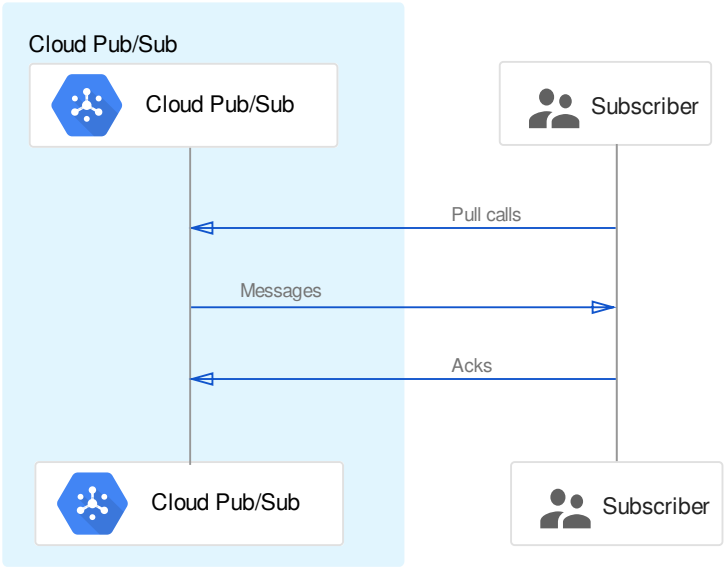
Typically, Pub/Sub delivers each message once and in the order in which it was published. However, messages may sometimes be delivered out of order or more than once. In general, accommodating

more-than-once delivery requires your subscriber to be idempotent (http://en.wikipedia.org/wiki/Idempotence#Computer_science_meaning) when processing messages. You can achieve exactly once processing of Pub/Sub message streams using the `Apache Beam programming model` (/dataflow/docs/concepts/beam-programming-model). The Apache Beam I/O connectors lets you interact with Cloud Dataflow (/dataflow/docs/concepts/streaming-with-cloud-pubsub) via controlled sources and sinks. You can use the Apache Beam `PubSubIO` connector (for Java (https://beam.apache.org/releases/javadoc/current/org/apache/beam/sdk/io/gcp/pubsub/PubsubIO.html) and Python (https://beam.apache.org/releases/pydoc/current/apache_beam.io.gcp.pubsub.html)) to read from Cloud Pub/Sub. You can also achieve ordered processing with Cloud Dataflow by using the standard sorting APIs of the service. Alternatively, to achieve ordering, the publisher of the topic to which you subscribe can include a sequence token in the message. See Message Ordering (/pubsub/docs/ordering) for more information.

A subscription can use either the pull or push mechanism for message delivery. You can change or configure the mechanism at any time.

In pull delivery, your subscriber application initiates requests to the Pub/Sub server to retrieve messages.

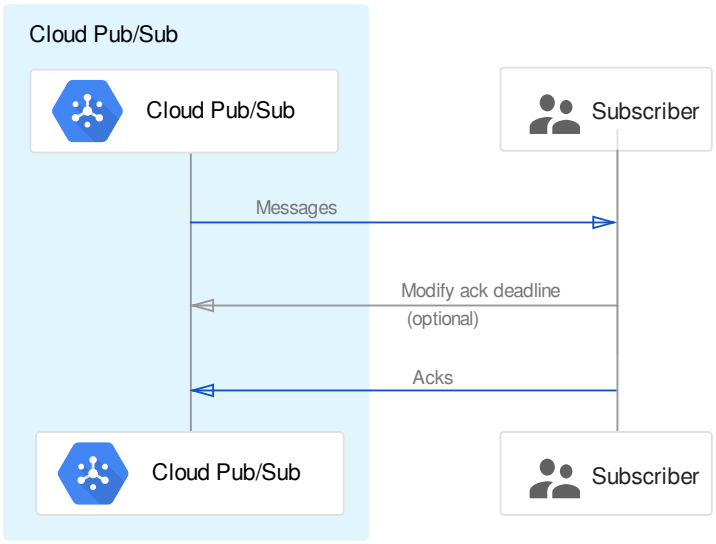
1. The subscribing application explicitly calls the pull method, which requests messages for delivery.
2. The Pub/Sub server responds with the message (or an error if the queue is empty) , and an ack ID.
3. The subscriber explicitly calls the acknowledge method, using the returned ack ID to acknowledge receipt.



In push delivery, Pub/Sub initiates requests to your subscriber application to deliver messages.

New push subscriptions cannot be created for any projects protected by VPC Service Controls (/vpc-service-controls
g push subscriptions will continue to function, although they will not be protected by VPC Service Controls. Contact
e Cloud administrator for more details.

1. The Pub/Sub server sends each message as an HTTPS request to the subscriber application at a pre-configured endpoint.
2. The endpoint acknowledges the message by returning an HTTP success status code. A non-success response indicates that the message should be resent.



Pub/Sub dynamically adjusts the rate of push requests based on the rate at which it receives success responses.

The following table offers some guidance in choosing the appropriate delivery mechanism for your application:

Pull	Push
• Large volume of messages (many more than 1/second). • Efficiency and throughput of message processing is critical. • Public HTTPS endpoint, with non-self-signed SSL certificate, is not feasible to set up.	• Multiple topics that must be processed by the same webhook. • App Engine Standard and Cloud Functions subscribers. • Environments where Google Cloud dependencies (such as credentials and the client library) are not feasible to set up.

The following table compares pull and push delivery:

	Pull	Push
Endpoints	Any device on the internet that has authorized credentials is able to call the Pub/Sub API.	An HTTPS server with non-self-signed certificate accessible on the public web. The receiving endpoint may be decoupled from the Pub/Sub subscription, so that messages from multiple subscriptions may be sent to a single endpoint.
Load	Multiple subscribers can make pull calls to the same	The push endpoint can be a load balancer.

balancing	"shared" subscription. Each subscriber will receive a subset of the messages.	
Configuration	No configuration is necessary.	No configuration is necessary for App Engine apps in the same project as the subscriber. Configuration (and verification) of push endpoints is required in the Google Cloud Console for all other endpoints. Endpoints must be reachable via DNS names and have SSL certificates installed.
Flow control	The subscriber client controls the rate of delivery. The subscriber can dynamically modify the ack deadline, allowing message processing to be arbitrarily long.	The Pub/Sub server automatically implements flow control. There is no need to handle message flow at the client side, although it is possible to indicate that the client cannot handle the current message load by passing back an HTTP error.
Efficiency and throughput	Achieves high throughput at low CPU and bandwidth by allowing batched delivery and acknowledgments as well as massively parallel consumption. May be inefficient if aggressive polling is used to minimize message delivery time.	Delivers one message per request and limits maximum number of outstanding messages.

By default, subscriptions expire after 31 days of inactivity (for instance, if there are no active connections, pull requests, or push successes). If Pub/Sub detects subscriber activity, the subscription deletion clock restarts. Using subscription expiration policies, you can configure the inactivity duration or make the subscription persistent regardless of activity. You can also delete a subscription manually.

Note that although you can create a new subscription with the same name as a deleted one, the new subscription has no relationship to the old one. Even if the deleted subscription had a large number of unacknowledged messages, a new identically-named subscription would have no backlog (no messages waiting for delivery) at the time it is created.

For more information about working with subscriptions, see Configuring subscriptions (/pubsub/docs/admin#configuring_subscriptions).