This document provides information about publishing messages.

- To learn about creating, deleting, and administering topics and subscriptions, see Managing Topics and Subscriptions (/pubsub/docs/admin).
- To restrict the locations in which message data is stored on a per-topic basis, see Restricting Pub/Sub resource locations (/pubsub/docs/resource-location-restriction).
- To learn more about receiving messages, see the Subscriber Guide (/pubsub/docs/subscriber).

A publisher application creates and sends messages to a **topic**. Pub/Sub offers at-least-once message delivery (/pubsub/docs/subscriber) and best-effort ordering to existing subscribers, as explained in the Subscriber Overview (/pubsub/docs/subscriber).

The general flow for a publisher application is:

1. Create a message containing your data.
2. Send a request to the Pub/Sub Server to publish the message to the desired topic.

See the Client Libraries Getting Started Guide (/pubsub/docs/reference/libraries) to set up your environment in the programming language of your choice.

When using JSON over REST, message data must be base64-encoded. The entire request including one or more messages must be smaller than 10MB, after decoding. Note that the message payload must not be empty; it must contain either a non-empty data field, or at least one attribute.

Client libraries, depending on your choice of programming language, can publish messages synchronously or asynchronously. Asynchronous publishing allows for batching and higher

throughput in your application.

All client libraries support publishing messages asynchronously. See the API Reference documentation (/pubsub/docs/reference/libraries#additional_resources) for your chosen programming language to see if its client library also supports publishing messages synchronously, if that is your preferred option.

A server-generated ID (unique within the topic) is returned on the successful publication of a message.

You can embed custom attributes as metadata in Pub/Sub messages. Attributes can be text strings or byte strings. The message schema can be represented as follows:

The `PubsubMessage` JSON schema is published as part of the REST (/pubsub/docs/reference/rest/v1/PubsubMessage) and RPC (/pubsub/docs/reference/rpc/google.pubsub.v1#google.pubsub.v1.PubsubMessage) documentation.

The Pub/Sub client libraries batch multiple messages into a single call to the service. Larger batch sizes increase message throughput (rate of messages sent per CPU). The cost of batching is latency for individual messages, which are queued in memory until their corresponding batch is filled and ready to be sent over the network. To minimize latency, batching should be turned off. This is particularly important for applications that publish a single message as part of a request-response sequence. A common example of this pattern is encountered in serverless, event-driven applications using Cloud Functions or App Engine.

Messages can be batched based on request size (in bytes), number of messages, and time. You can override the default settings as shown in this sample:

Publishing failures are automatically retried, except for errors that do not warrant retries. This sample code demonstrates creating a publisher with custom retry settings (note that not all client libraries support custom retry settings; see the API Reference documentation (/pubsub/docs/reference/libraries#additional_resources) for your chosen language):

Retry settings control both the total number of retries and exponential backoff (how long the client waits between subsequent retries). The initial RPC timeout is the time the client waits for the initial RPC to succeed before retrying. The total timeout is the time the client waits before it stops retrying. To retry publish requests, the initial RPC timeout should be shorter than the total timeout.

Once the first RPC fails or times out, the exponential backoff logic determines when the subsequent retries occur. On each retry, the RPC timeout increases by this multiplier, up to the maximum RPC timeout. In addition, the retry delay settings determine how long the client waits between getting an error or timeout and initiating the next request.

Support for concurrency depends on your programming language. Refer to the API Reference documentation (/pubsub/docs/reference/libraries#additional_resources) for more information.

The following sample illustrates how to control concurrency in a publisher: