The Pub/Sub API exports metrics via Stackdriver (/monitoring/). Stackdriver allows you to create monitoring dashboards (/monitoring/charts/) and alerts (/monitoring/alerts/) or access the metrics programmatically.

Go to Stackdriver (https://app.google.stackdriver.com/) in Google Cloud Console to view Stackdriver monitoring dashboards or to define Stackdriver alerts. You can also use the Stackdriver monitoring API (/monitoring/api/v3) to query and view metrics for subscriptions and topics.

- To see the usage metrics that Pub/Sub reports to Stackdriver, view the Metrics List (/monitoring/api/metrics_gcp#gcp-pubsub) in the Stackdriver documentation.
- To see the details for the `pubsub_topic` (/monitoring/api/resources#tag_pubsub_topic) `pubsub_subscription` (/monitoring/api/resources#tag_pubsub_subscription) or `pubsub_snapshot` (/monitoring/api/resources#tag_pubsub_snapshot) monitored resource types, view Monitored Resource Types (/monitoring/api/resources) in the Stackdriver documentation.

You can use the APIs and services quotas dashboard (https://console.cloud.google.com/apis/api/pubsub.googleapis.com/quotas?project=_) to monitor the current utilization for a given topic or subscription.

Those metrics are:

- `pubsub.googleapis.com/topic/byte_cost`

- `pubsub.googleapis.com/subscription/byte_cost`

Note that these metrics are in bytes, whereas quota is measured in kilobytes.

To ensure that your subscribers are keeping up with the flow of messages, create a dashboard that shows the following metrics, aggregated by resource, for all your subscriptions.:

- `subscription/num_undelivered_messages`
- `subscription/oldest_unacked_message_age`

Create alerts that will fire when these values are unusually large in the context of your system. For instance, the absolute number of undelivered messages is not necessarily meaningful. A backlog of a million messages might be acceptable for a million message-per-second subscription, but unacceptable for a one message-per-second subscription.

Symptoms	Problem	Solutions
Both the **`oldest_unacked_message_age`** and **`num_undelivered_messages`** are growing in tandem.	Subscribers not keeping up with message volume	• Add more threads o • Add more machines containers • Look for s bugs in yo that preve successfu acknowle messages processin timely fas Monitorin deadline e (#monito
If there is a steady, small backlog size combined with a steadily growing	Stuck	Examine your

`oldest_unacked_message_age`, there may be a small number of messages that cannot be processed.	messages	logs to under whether som are causing y crash. It's unl possible —tha offending me stuck on Pub than in your c a (/pubsub/dc support case are confident successfully p each messag
The `oldest_unacked_message_age` exceeds the subscription's message retention duration (/pubsub/docs/admin#retaining_unacknowledged_and_acknowledged_messages).	Permanent data loss	Set up an aler well in advan subscription's retention dur lapsing.

In order to reduce end-to-end latency of message delivery, Pub/Sub allows subscriber clients a limited amount of time to acknowledge a given message (known as the "ack deadline") before re-delivering the message. If your subscribers take too long to acknowledge messages, the messages will be re-delivered, resulting in the subscribers seeing duplicate messages. This can happen for a number of reasons:

- Your subscribers are under-provisioned (you need more threads or machines).
- Each message takes longer to process than the message acknowledgement deadline. Google Cloud Client Libraries generally extend the deadline for individual messages up to a configurable maximum. However, a maximum extension deadline is also in effect for the libraries.
- Some messages consistently crash the client.

It can be useful to measure the rate at which subscribers miss the ack deadline. The specific metric depends on the subscription type:

- **Pull:** `subscription/pull_ack_message_operation_count` filtered by `response_code != "success"`
- **Streaming Pull:** `subscription/streaming_pull_ack_message_operation_count` filtered by `response_code != "success"`
- **Push:** `subscription/push_request_count` filtered by `response_code != "success"`

Excessive ack deadline expiration rates can result in costly inefficiencies in your system. You pay for every redelivery and for attempting to process each message repeatedly. Conversely, a small expiration rate (for example, 0.1-1%) might be healthy.

For push subscriptions, you should also monitor these metrics:

- `subscription/push_request_count`

  Group the metric by `response_code` and `subcription_id`. Since Pub/Sub push subscriptions use response codes as implicit message acknowledgements, it is important to monitor push request response codes. Because push subscriptions exponentially back off (/pubsub/docs/push#quotas_limits_and_delivery_rate) when they encounter timeouts or errors, your backlog can grow quickly based on how your endpoint responds.

  Consider setting an alert for high error rates (create a metric filtered by response class), since those rates lead to slower delivery and a growing backlog. However, push request counts are likely to be more useful as a tool for investigating growing backlog size and age.

- `subscription/num_outstanding_messages`

  Pub/Sub generally limits the number of outstanding messages (/pubsub/quotas). You should aim for fewer than 1000 outstanding messages in most situations. As a rule, the service adjusts the limit based on the overall throughput of the subscription in increments of 1000, once the throughput achieves a rate on the order of ten thousand messages per second. No specific guarantees are made beyond the maximum value, so 1000 is a good guide.

- `subscription/push_request_latencies`

  This metric helps you understand your push endpoint's response latency distribution. Because of the limit on the number of outstanding messages, endpoint latency affects subscription throughput. If it takes 100 milliseconds to process each message, your throughput limit is likely to be 10 messages per second.

The primary goal of a publisher is to persist message data quickly. Monitor this performance using `topic/send_request_count`, grouped by `response_code`. This metric gives you an indication of whether Pub/Sub is healthy and accepting requests.

A background rate of retryable errors (significantly lower than 1%) should not be a cause for concern, since most Google Cloud Client Libraries retry message failures. You should investigate error rates that are greater than 1%. Because non-retryable codes are handled by your application (rather than the client library), you should examine response codes. If your publisher application does not have a good way of signaling an unhealthy state, consider setting an alert on the `send_request_count` metric.

It is equally important to track failed publish requests in your publish client. While client libraries generally retry failed requests, they do not guarantee publication. Refer to Publishing messages (/pubsub/docs/publisher#retry) for ways to detect permanent publish failures when using Google Cloud Client Libraries. At a minimum, your publisher application should log permanent publish errors. If you log those errors to Stackdriver Logging, you can set up a logs-based metric (/logging/docs/logs-based-metrics/) with an alert.