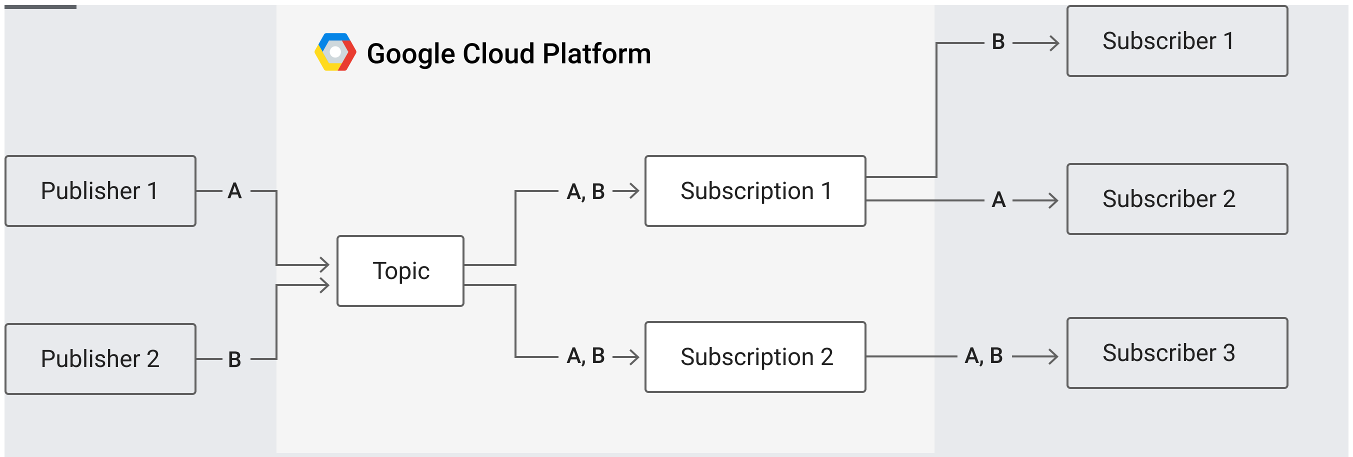
Pub/Sub (https://cloud.google.com/pubsub/docs) is an asynchronous messaging service designed to be highly reliable and scalable. The service is built on a core Google infrastructure component that many Google products have relied upon for over a decade. Google products including Ads, Search and Gmail use this infrastructure to send over 500 million messages per second, totaling over 1TB/s of data. This article describes the salient design features that enables Pub/Sub to provide this type of scale reliably.

Pub/Sub is a *publish/subscribe* (*Pub/Sub*) *service*: a messaging service where the senders of messages are decoupled from the receivers of messages. There are several key concepts in a Pub/Sub service:

- *Message*: the data that moves through the service.
- *Topic*: a named entity that represents a feed of messages.
- *Subscription*: a named entity that represents an interest in receiving messages on a particular topic.
- *Publisher* (also called a producer): creates messages and sends (publishes) them to the messaging service on a specified topic.
- *Subscriber* (also called a consumer): receives messages on a specified subscription.

The basic flow of messages through Pub/Sub can be summarized in the following diagram:



In this scenario, there are two publishers publishing messages on a single topic. There are two subscriptions to the topic. The first subscription has two subscribers, meaning messages will be load-balanced across them, with each subscriber receiving a subset of the messages. The second subscription has one subscriber that will receive all of the messages. The bold letters represent messages. Message A comes from Publisher 1 and is sent to Subscriber 2 via Subscription 1, and to Subscriber 3 via Subscription 2. Message B comes from Publisher 2 and is sent to Subscriber 1 via Subscription 1 and to Subscriber 3 via Subscription 2.

A messaging service like Pub/Sub can be judged on its performance in three aspects: scalability, availability, and latency. These three factors are often at odds with each other, requiring compromises on one to improve the other two.

The terms "scalability," “availability,” and “latency” can refer to different properties of a system, so the following sections describe how they are defined in Pub/Sub.

A scalable service should be able to handle increases in load without noticeable degradation of latency or availability. "Load" can refer to various dimensions of usage in Pub/Sub:

- Number of topics
- Number of publishers
- Number of subscriptions

- Number of subscribers
- Number of messages
- Size of messages
- Rate of messages (throughput) published or consumed
- Size of backlog on any given subscription

In a distributed system, the types and severity of problems can vary greatly. A system's availability is measured on how well it deals with different types of issues, gracefully failing over in a way that is unnoticeable to end users. Failures can occur in hardware (e.g., disk drives not working (https://status.cloud.google.com/incident/compute/15056#5719570367119360) or network connectivity problems (http://www.slate.com/blogs/future_tense/2014/08/15/shark_attacks_threaten_google_s_undersea_internet_cables_video.html)
), in software, and due to load. Failure due to load could happen when a sudden increase in traffic in the service (or in other software components running on the same hardware or in software dependencies) results in resource scarcity. Availability can also degrade due to human error, where one makes mistakes in building or deploying software or configurations.

Latency is a time-based measure of the performance of a system. A service generally wants to minimize latency wherever possible. For Pub/Sub, the two most important latency metrics are:

1. The amount of time it takes to acknowledge a published message.
2. The amount of time it takes to deliver a published message to a subscriber.

This section explains the design of Pub/Sub to show how the service attains its scalability and low latency while retaining availability. The system is designed to be *horizontally scalable*, where an increase in the number of topics, subscriptions, or messages can be handled by increasing the number of instances of running servers.

Pub/Sub servers run in most GCP regions (/about/locations/) around the world. This allows the service to offer fast, global data access, while giving users control over where messages are stored. Cloud Pub/Sub offers global data access in that publisher and subscriber clients are not aware of the location of the servers to which they connect or how those services route the data.

Pub/Sub's load balancing mechanisms direct publisher traffic to the nearest GCP data center where data storage is allowed, as defined in the Resource Location Restriction (/pubsub/docs/resource-location-restriction) section of the IAM & admin console (https://console.cloud.google.com/iam-admin/orgpolicies/list?project=_&). This means that publishers in multiple regions may publish messages to a single topic with low latency. Any individual message is stored in a single region. However, a topic may have messages stored in many regions. When a subscriber client requests messages published to this topic, it connects to the nearest server which aggregates data from all messages published to the topic for delivery to the client.

Pub/Sub is divided into two primary parts: the *data plane*, which handles moving messages between publishers and subscribers, and the *control plane*, which handles the assignment of publishers and subscribers to servers on the data plane. The servers in the data plane are called *forwarders*, and the servers in the control plane are called *routers*. When publishers and subscribers are connected to their assigned forwarders, they do not need any information from the routers (as long as those forwarders remain accessible). Therefore, it is possible to upgrade the control plane of Pub/Sub without affecting any clients that are already connected and sending or receiving messages.

The Pub/Sub control plane distributes clients to forwarders in a way that provides scalability, availability, and low latency for all clients. Any forwarder is capable of serving clients for any topic or subscription. When a client connects to Pub/Sub, the router decides the data centers the client should connect to based on shortest *network distance*, a measure of the latency on the connection between two points. Within any given data center the router tries to distribute overall load across the set of available forwarders. The router must balance two different goals when performing this assignment: (a) uniformity of load (i.e., ideally every forwarder is equally loaded); and (b) stability of assignments (i.e., ideally a change in load or a change in the set of available forwarders changes the smallest number of existing assignments). The router uses a variant of consistent hashing (https://research.googleblog.com/2017/04/consistent-hashing-with-bounded-loads.html) developed by Google Research (https://research.google.com/) to achieve a tunable balance between consistency and uniformity. The router provides the client with an ordered list of forwarders it can consider connecting to. This ordered list may change based on forwarder availability and the shape of the load from the client.

A client takes this list of forwarders and connects to one or more of them. The client prefers connecting to the forwarders most recommended by the router, but also takes into consideration any failures that have occurred, e.g., it may decide to try forwarders in a different data center if several attempts to the nearest ones have failed. In order to abstract Pub/Sub clients away from these implementation details, there is a service proxy between the clients and forwarders that performs this connection optimization on behalf of clients.

The data plane receives messages from publishers and sends them to clients. Perhaps the best way of understanding Pub/Sub's data plane is by looking at the life of a message, from the moment it is received by the service to the moment it is no longer present in the service. Let us trace the steps of processing a message. For the purposes of this section, we assume that the topic on which the message is published has at least one subscription attached to it. In general, a message goes through these steps:

1. A publisher sends a message.
2. The message is written to storage.
3. Pub/Sub sends an acknowledgement to the publisher that it has received the message and guarantees its delivery to all attached subscriptions.
4. At the same time as writing the message to storage, Pub/Sub delivers it to subscribers.
5. Subscribers send an acknowledgement to Pub/Sub that they have processed the message.
6. Once at least one subscriber for each subscription has acknowledged the message, Pub/Sub deletes the message from storage.

First, a publisher sends a message on a topic to Pub/Sub. It is encrypted by the proxy layer and sent to a publishing forwarder, a forwarder to which the publisher is connected. In order to ensure delivery, the message is immediately written to storage. The forwarder initially writes the message to N clusters (where N is an odd number) and considers the message persisted when it has been written to at least $\lceil N/2 \rceil$ clusters. Once a message is persisted, the publishing forwarder acknowledges receipt of the message back to the publisher, at which point Pub/Sub guarantees that the message will be delivered to all attached subscriptions. A background process regularly writes any messages that are not in all N clusters to the clusters missing the messages.

Within each cluster, the message is written to M independent disks (where M is an odd number), requiring the data to be on ⌈M/2⌉ disks before it is considered persisted in that cluster. In total, any message published will be written to at least ⌈M/2⌉ independent disks in ⌈N/2⌉ clusters before it is considered persisted and will eventually be replicated to N*M disks.

The publishing forwarder has a list of all subscriptions that are attached to a topic. It is responsible for persisting both the published messages and the metadata describing which messages have been acknowledged by each subscription. The set of messages received and stored by a publishing forwarder for a particular topic, along with this tracking of acknowledged messages, is called a "publish message source." Depending on the throughput requirements for the topic, a single publisher may send its messages to multiple publishing forwarders and store messages in multiple publish message sources. Different publishers for the same topic may also send messages to different publishing forwarders. Each message is sent to only a single publishing forwarder. Pub/Sub dynamically tunes the number of publishing forwarders that receive messages for a particular topic as the throughput changes.

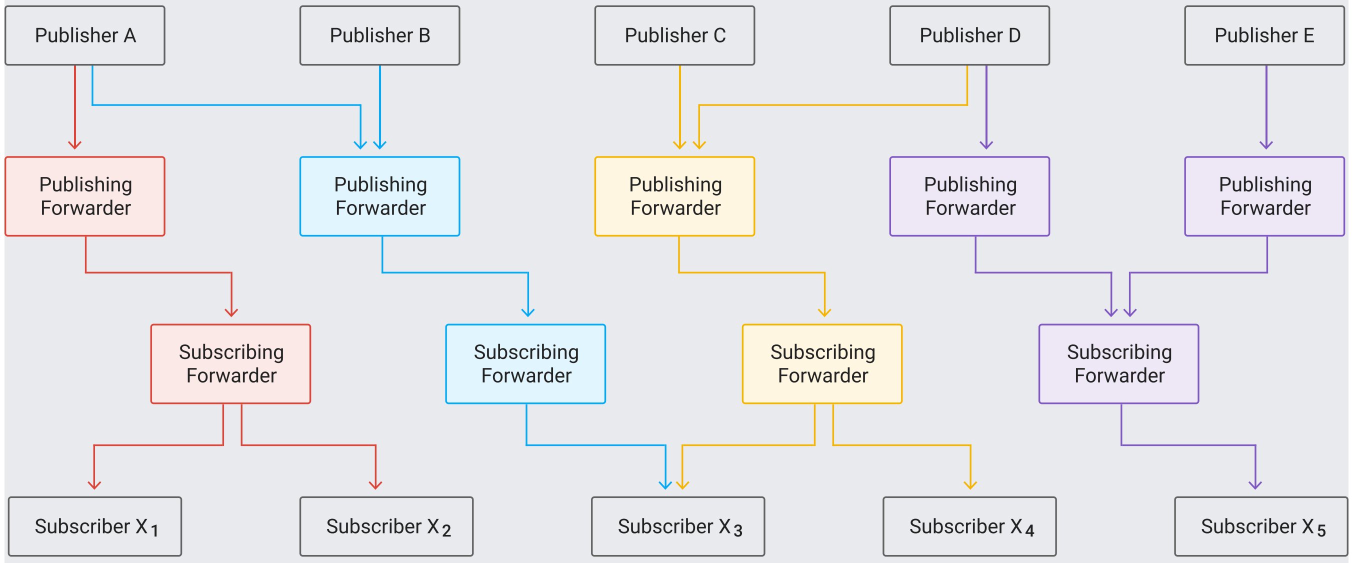
Subscribers receive messages by connecting to subscribing forwarders, forwarders through which messages flow to subscribers from publishers. “Connecting” in the case of a pull subscriber means issuing a pull request. “Connecting” in the case of a push subscriber means having the push endpoint registered with Pub/Sub. Once a subscription is created, it is guaranteed that any messages published after that point will be delivered to that subscription, what we call a sync-point guarantee.

Each subscribing forwarder needs to request messages from publishing forwarders that have publish message sources for the topic. Like publishers, subscribers may connect to more than one subscribing forwarder in order to receive messages. That way, not every subscribing forwarder needs to be aware of or receive messages from every publish message source for a topic–an important property for Pub/Sub to be able to scale horizontally. Based on the throughput of messages being delivered to subscribers, Pub/Sub dynamically tunes the number of subscribing forwarders through which subscribers receive messages for a particular topic as the throughput changes.

A subscribing forwarder makes requests to one or more publishing forwarders that have publish message sources for a topic to ask for the messages it needs. The publishing forwarder sends the unacknowledged messages to the subscribing forwarder, which then relays the messages to a subscriber.

Once a subscriber processes a message, it sends an acknowledgement back to the subscribing forwarder. The subscribing forwarder relays this acknowledgement to the publishing forwarder, which stores the acknowledgement in the publish message source. Once all subscriptions on a topic have acknowledged a message, the message is asynchronously deleted from the publish message source and from storage.

Different messages for a single topic and subscription can flow through many publishers, subscribers, publishing forwarders, and subscribing forwarders. Publishers can publish to multiple forwarders simultaneously and subscribers may connect to multiple subscribing forwarders to receive messages. Therefore, the flow of messages through connections among publishers, subscribers, and forwarders can be complex. The following diagram shows how messages could flow for a single topic and subscription, where different colors indicate the different paths messages may take from publishers to subscribers:



Ensuring that a distributed system like Pub/Sub can stay up and running and effectively serve all customers requires a great deal of visibility into and control of the system. Maintaining the service is the responsibility of our Site Reliability Engineers (https://books.google.com/books?id=81UrjwEACAAJ) (SREs). For Pub/Sub, these engineers are based in multiple locations around the world in order to provide 24/7 coverage.

The first part of maintaining a system like Pub/Sub is to have the ability to test software before it is used by customers. In order to make that possible, there are three Pub/Sub environments: test, staging, and production. Test and staging do not contain any customer traffic; they contain only our continuously-running tests and monitoring that help to find any issues with releases. These environments receive new releases of the software before production. The difference between test and staging is that the latter is an exact replica of what is in (or will very shortly be in) the production environment, including software version and command-line flags. The former may have features enabled that developers are currently working on and plan for release sometime in the future.

The procedure for rolling out and testing Pub/Sub is designed to minimize potential impact. Let's look at the typical steps for the rollout of a new version of Pub/Sub:

1. Ensure all unit tests and integration tests pass.
2. Build a new version of all the servers.
3. Deploy the new servers to the test and staging environments.
4. Run the servers on the test and staging environment for several days.
5. If there are no known issues, release servers to canary, a subset of the production environment that has a small amount of customer traffic.
6. If no problems are detected in canary, progressively roll the servers out to more of production over several days until they are released everywhere.

Since Pub/Sub is designed to be resilient to failures, e.g., through the separation of the control plane and data plane, rollouts of new versions of servers are seamless to customers and should have no impact on the performance they see.

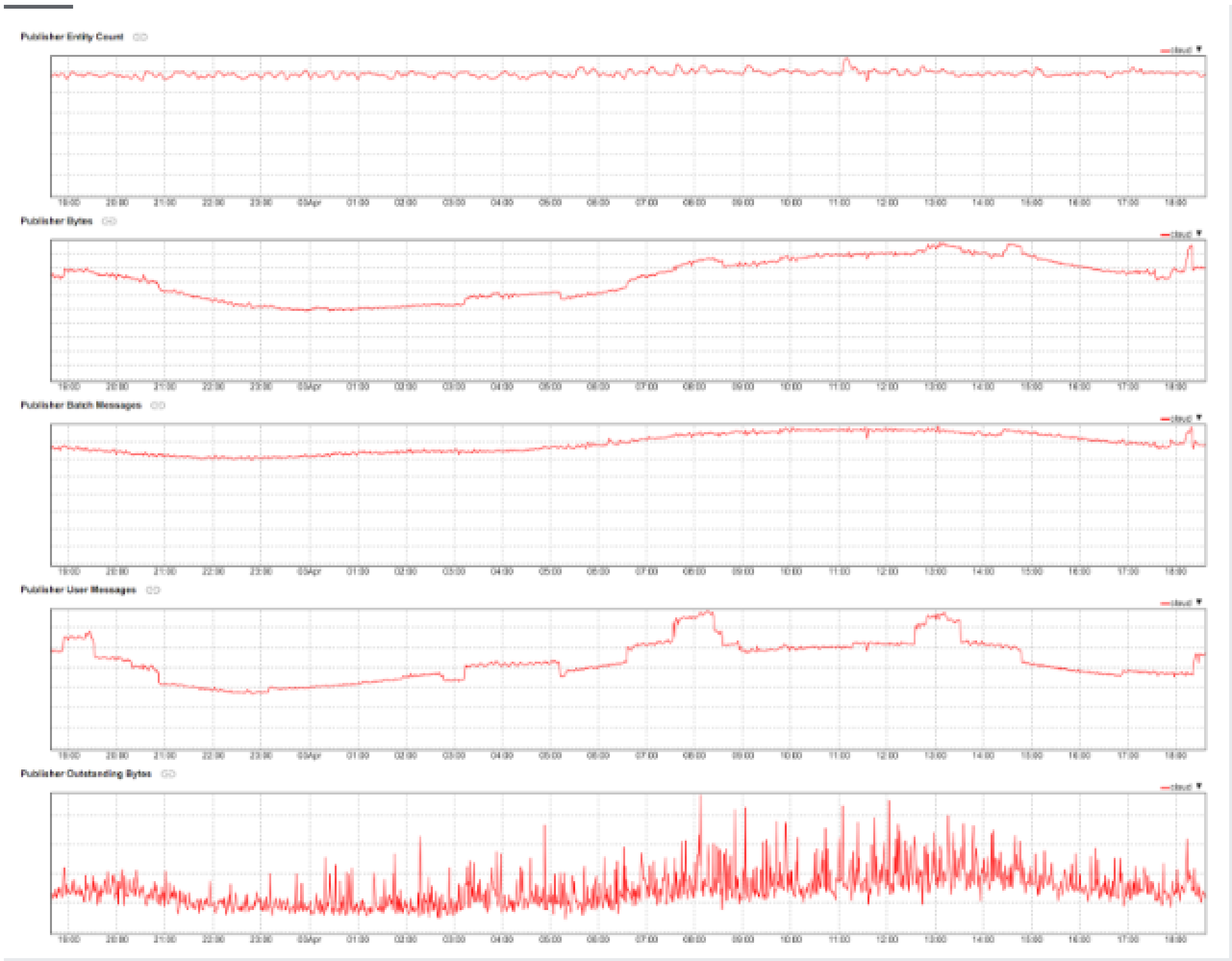
The key to keeping Pub/Sub up and running is to automatically detect and mitigate issues before they become visible to customers. Accomplishing this requires extensive monitoring of the system. The SREs maintain a set of *service level indicators (SLIs)*, well-defined metrics that describe the behavior of the system. Metrics may include "amount of time it takes for a CreateSubscription request to complete" or “rate of errors generated by Publish requests.” These metrics are measured in a variety of ways. Some of them are strictly internal to our forwarders and routers. For example, they measure how long it takes to write messages to disk. In total, there are ten SLIs and hundreds of additional metrics used to monitor the health of Pub/Sub.

All of these measures help to define internal *service level objectives (SLOs)*, specific targets for the SLIs. For example, "a CreateSubscription request should take no more than five seconds to complete." SREs are alerted for SLO violations, and must attend to alerts within five minutes.

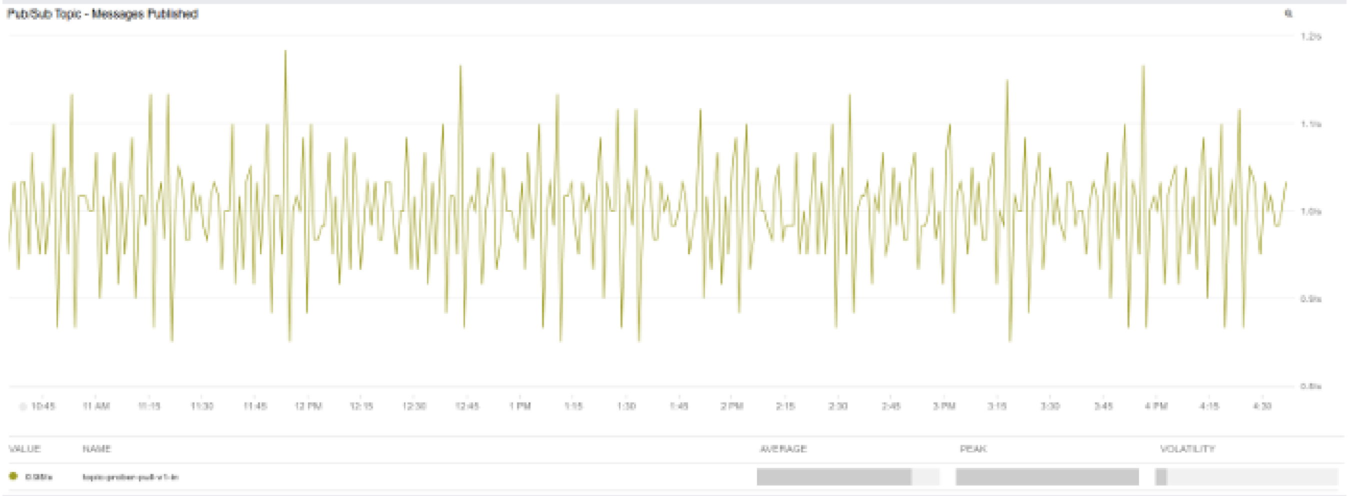
A *service level agreement (SLA)* lists the SLOs that define our performance guarantees to our customers and the consequences if we do not meet them. You can read Pub/Sub’s SLA (https://cloud.google.com/pubsub/sla).

We maintain a set of tasks that act as clients and predictably publish and subscribe called *probers*. Probers exist for both the data plane and the control plane. Each of our ten types of probers performs specific actions just as a customer would and measures how long the operations take. For instance, we have a prober that creates a new subscription, publishes a message, and sees how long it took to both create the subscription and receive the message. If the probers determine that any of thirty measured metrics are not what is expected, SREs are alerted.

The metrics for our servers and probers are summarized on several internal dashboards, the first place SREs look whenever diagnosing potential issues. These pages provide quick access to stats and graphs of the entire service, as pictured below. They can also be broken down by topic, data center, or individual task.



The metrics that are most interesting to users of the service are exposed via Google Cloud Monitoring (/monitoring/). In fact, even our own probers have charts available like this:



We have at our disposal several controls to help tune the performance of Pub/Sub. Some of these controls are designed to help with data center or machine outages. We can place *routing constraints* on some or all topics, which are rules specifying sets of clients that can and/or cannot connect to sets of forwarders. We use routing constraints to drain traffic away from individual forwarder tasks or whole data centers that are not functioning as expected.

Another tunable feature we have is flow control. This feature allows us to maximize throughput while preventing overload in the service. Flow control is a form of traffic shaping whereby sudden unexpected spikes in load can be smoothed out over time for greater service stability. Flow control operates system-wide or on a per-topic or per-subscriber basis to limit the number of messages or the number of bytes that are transferred or are outstanding. In this case "outstanding" means delivered to the client, but not yet acknowledged. Both flow control and routing constraints allow us to optimize the Pub/Sub's performance without customers having to worry about these low-level details.

The advantages in scalability, availability, and latency of a service like Pub/Sub define the value proposition for customers who are considering a move to managed cloud services. Any asynchronous messaging service has to be built from the ground up with these features in mind. With over a decade of experience in reliably delivering lots of messages quickly, the Pub/Sub team has built and maintains a service that can keep up with the demands of the most fundamental products at Google. Now that same service is available to all external customers who want to send their messages around the world without having to worry whether or not their messaging system can handle 2x, 10x, or 100x their current load.

Term	Description
cluster	A logical grouping of machines that generally share the same failure domain (e.g., shared local network and shared power).
control plane	The layer of Pub/Sub that handles the assignment of publishers and subscribers to servers on the data plane.

data plane	The layer of Pub/Sub that handles moving messages between publishers and subscribers.
forwarder	A server in the data plane.
global data access	Pub/Sub publisher and subscriber clients are unaware of the location of the data. All routing and storage is done by the service itself, in accordance with Location Restriction policy.
horizontally scalable	The ability of a service to seamlessly handle more load by increasing the number of instances of components of the service.
message	The data that moves through Pub/Sub.
network distance	A measure of the latency on the connection between two points.
prober	A task that acts as a client and predictably performs one or more actions on the Pub/Sub servers.
publish message source	A set of messages received and stored by a publishing forwarder and the set of IDs of messages acknowledged by all attached subscriptions.
publish/subscribe (Pub/Sub) service	A messaging service where the senders of messages are decoupled from the receivers of messages
publisher	A client of Pub/Sub that creates messages and sends (publishes) them on a specified topic.
router	A server in the control plane.
routing constraints	A list of rules indicating which forwarders should or should not be sent by routers to clients as possible endpoints to connect to.
service level agreement (SLA)	A list of SLOs that define a system's performance guarantees to customers and outlines the consequences if they are not met.
service level indicator (SLI)	A well-defined metric that describes the behavior of the system.
service level objective (SLO)	A specific target for a service level indicator.
subscriber	A client of Pub/Sub that receives messages on a specified subscription.
subscription	A named entity that represents an interest in receiving all messages on a particular topic.
sync-point guarantee	The time at which a subscriber is created, where all subsequent messages published will be delivered to that subscriber.
topic	A named entity that represents a feed of messages.