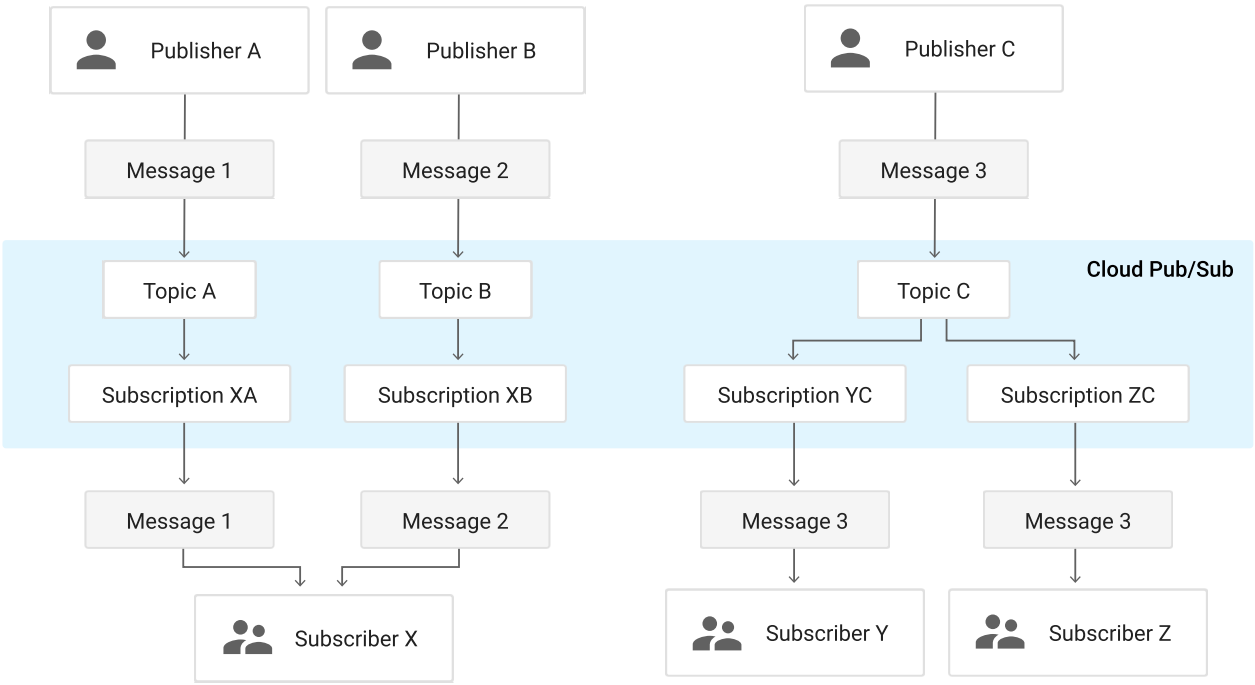
Pub/Sub brings the flexibility and reliability of enterprise message-oriented middleware to the cloud. At the same time, Pub/Sub is a scalable, durable event ingestion and delivery system that serves as a foundation for modern stream analytics pipelines. By providing many-to-many, asynchronous messaging that decouples senders and receivers, it allows for secure and highly available communication among independently written applications. Pub/Sub delivers low-latency, durable messaging that helps developers quickly integrate systems hosted on the Google Cloud Platform and externally.

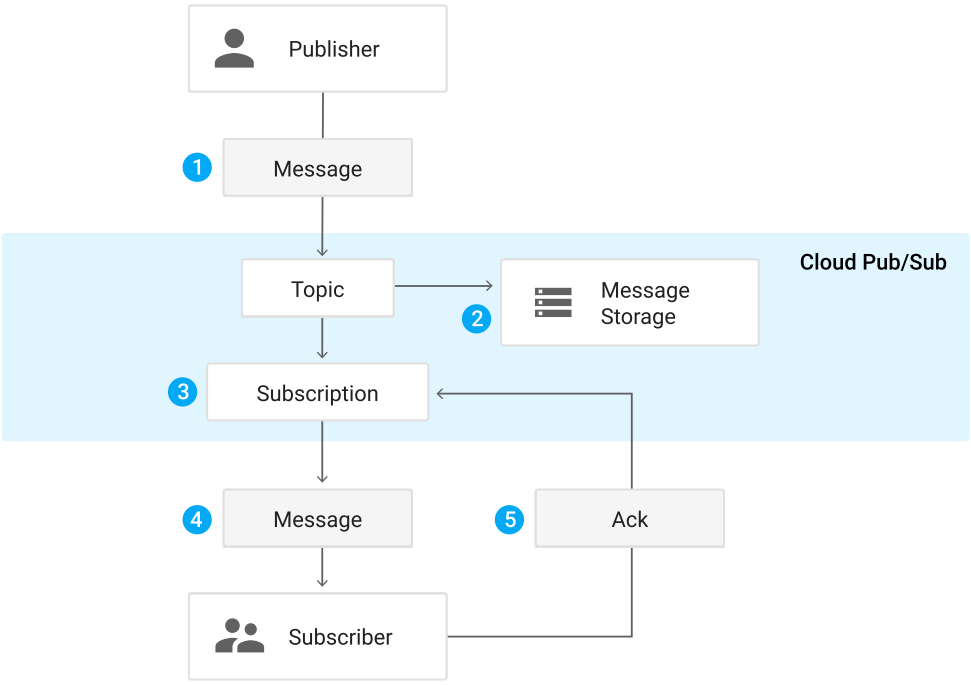
ou an experienced messaging developer and want to get started right away? Try the Quickstart sub/docs/quickstart-console). For a more comprehensive introduction, see Building a functioning Pub/Su m (/pubsub/docs/quickstart-py-mac). Other popular starting points are the Pub/Sub documentation overv sub/docs/), and Client Libraries (/pubsub/libraries).

- **Topic**: A named resource to which messages are sent by publishers.
- **Subscription**: A named resource representing the stream of messages from a single, specific topic, to be delivered to the subscribing application. For more details about subscriptions and message delivery semantics, see the Subscriber Guide (/pubsub/subscriber).
- **Message**: The combination of data and (optional) attributes that a publisher sends to a topic and is eventually delivered to subscribers.
- **Message attribute**: A key-value pair that a publisher can define for a message. For example, key `iana.org/language_tag` and value `en` could be added to messages to mark them as readable by an English-speaking subscriber.

A publisher application creates and sends messages to a *topic*. Subscriber applications create a *subscription* to a topic to receive messages from it. Communication can be one-to-many (fan-out), many-to-one (fan-in), and many-to-many.

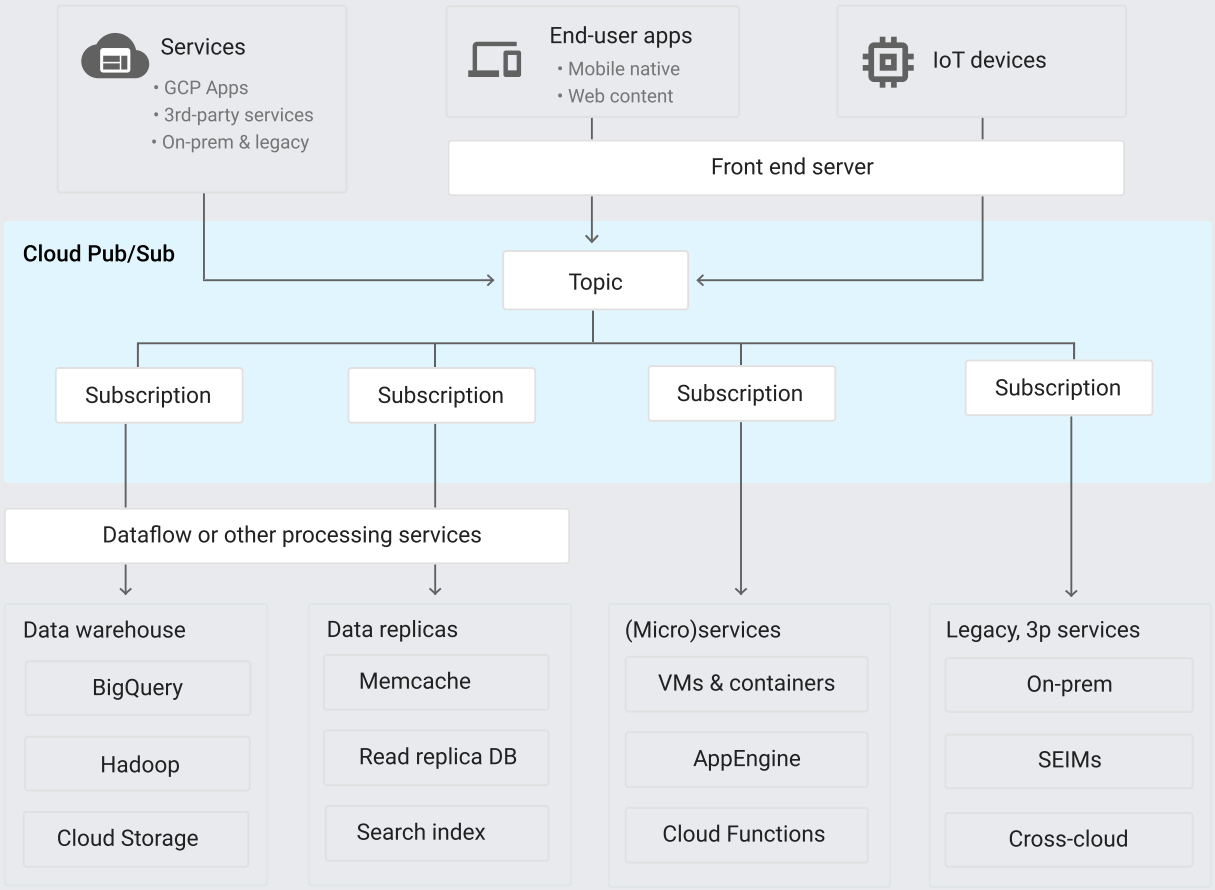


Here is an overview of the components in the Pub/Sub system and how messages flow between them:



1. A *publisher* application (#endpoints) creates a *topic* in the Pub/Sub service and sends *messages* to the topic. A message contains a payload and optional *attributes* that describe the payload content.
2. The service ensures that published messages are retained on behalf of subscriptions. A published message is retained for a subscription until it is acknowledged by any subscriber consuming messages from that subscription.
3. Pub/Sub forwards messages from a topic to all of its subscriptions, individually. Each subscription receives messages either by Pub/Sub *pushing* them to the subscriber's chosen endpoint, or by the subscriber *pulling* them from the service.
4. The subscriber receives pending messages from its subscription and acknowledges each one to the Pub/Sub service.
5. When a message is acknowledged by the subscriber, it is removed from the subscription's message queue.

Publishers can be any application that can make HTTPS requests to googleapis.com: an App Engine app, a web service hosted on Google Compute Engine or any other third-party network, an installed app for desktop or mobile device, or even a browser.



Pull subscribers can also be any application that can make HTTPS requests to googleapis.com.

Push subscribers must be Webhook endpoints that can accept POST requests over HTTPS.

- **Balancing workloads in network clusters.** For example, a large queue of tasks can be efficiently distributed among multiple workers, such as Google Compute Engine instances.
- **Implementing asynchronous workflows.** For example, an order processing application can place an order on a topic, from which it can be processed by one or more workers.
- **Distributing event notifications.** For example, a service that accepts user signups can send notifications whenever a new user registers, and downstream services can subscribe to receive notifications of the event.
- **Refreshing distributed caches.** For example, an application can publish invalidation events to update the IDs of objects that have changed.

- **Logging to multiple systems.** For example, a Google Compute Engine instance can write logs to the monitoring system, to a database for later querying, and so on.
- **Data streaming from various processes or devices.** For example, a residential sensor can stream data to backend servers hosted in the cloud.
- **Reliability improvement.** For example, a single-zone Compute Engine service can operate in additional zones by subscribing to a common topic, to recover from failures in a zone or region.