What is Pub/Sub?

Key Point: Understand what is Pub/Sub, why do businesses require Pub/Sub, and the advantages of Pub/Sub compared to similar technologies. Also, learn about core Pub/Sub concepts that include the terms topic, publisher, and subscriber.

Pub/Sub allows services to communicate asynchronously, with latencies on the order of 100 milliseconds.

Pub/Sub is used for streaming analytics and data integration pipelines to ingest and distribute data. It's equally effective as a messaging-oriented middleware for service integration or as a queue to parallelize tasks.

Pub/Sub enables you to create systems of event producers and consumers, called **publishers** and **subscribers**. Publishers communicate with subscribers asynchronously by broadcasting events, rather than by synchronous remote procedure calls (RPCs).

Publishers send events to the Pub/Sub service, without regard to how or when these events are to be processed. Pub/Sub then delivers events to all the services that react to them. In systems communicating through RPCs, publishers must wait for subscribers to receive the data. However, the asynchronous integration in Pub/Sub increases the flexibility and robustness of the overall system.

To get started with Pub/Sub, check out the Quickstart using Google Cloud console (/pubsub/docs/create-topic-console). For a more comprehensive introduction, see Building a Pub/Sub messaging system (/pubsub/docs/quickstart-py-mac).

Common use cases

- **Ingestion user interaction and server events.** To use user interaction events from end-user apps or server events from your system, you might forward them to Pub/Sub. You can then use a stream processing tool, such as Dataflow, which delivers the events to databases. Examples of such databases are BigQuery, Cloud Bigtable, and Cloud Storage. Pub/Sub lets you gather events from many clients simultaneously.
- **Real-time event distribution.** Events, raw or processed, may be made available to multiple applications across your team and organization for real- time processing.

Pub/Sub supports an "enterprise event bus" and event-driven application design patterns. Pub/Sub lets you integrate with many Google systems that export events to Pub/Sub.

- **Replicating data among databases.** Pub/Sub is commonly used to distribute change events from databases. These events can be used to construct a view of the database state and state history in BigQuery and other data storage systems.
- **Parallel processing and workflows.** You can efficiently distribute many tasks among multiple workers by using Pub/Sub messages to connect to Cloud Functions. Examples of such tasks are compressing text files, sending email notifications, evaluating AI models, and reformatting images.
- **Enterprise event bus.** You can create an enterprise-wide real-time data sharing bus, distributing business events, database updates, and analytics events across your organization.
- **Data streaming from applications, services, or IoT devices.** For example, a SaaS application can publish a real-time feed of events. Or, a residential sensor can stream data to Pub/Sub for use in other Google Cloud products through a Dataflow pipeline.
- **Refreshing distributed caches.** For example, an application can publish invalidation events to update the IDs of objects that have changed.
- **Load balancing for reliability.** For example, instances of a service may be deployed on Compute Engine in multiple zones but subscribe to a common topic. When the service fails in any zone, the others can pick up the load automatically.

Types of Pub/Sub services

Pub/Sub consists of two services:

- **Pub/Sub service.** This messaging service is the default choice for most users and applications. It offers the highest reliability and largest set of integrations, along with automatic capacity management. Pub/Sub guarantees synchronous replication of all data to at least two zones and best-effort replication to a third additional zone.
- **Pub/Sub Lite service.** A separate but similar messaging service built for lower cost. It offers lower reliability compared to Pub/Sub. It offers either zonal or regional topic storage. Zonal Lite topics are stored in only one zone. Regional Lite topics replicate data to a second zone asynchronously. Also, Pub/Sub Lite requires you to pre-provision and manage storage and throughput capacity. Consider Pub/Sub Lite only

for applications where achieving a low cost justifies some additional operational work and lower reliability.

For more details about the differences between Pub/Sub and Pub/Sub Lite, see Choosing Pub/Sub or Pub/Sub Lite (/pubsub/docs/choosing-pubsub-or-lite).

Comparing Pub/Sub to other messaging technologies

Pub/Sub combines the horizontal scalability of Apache Kafka (/learn/what-is-apache-kafka) and Pulsar (https://pulsar.apache.org/docs/en/2.4.0/concepts-overview/) with features found in traditional messaging middleware such as Apache ActiveMQ and RabbitMQ. Examples of such features are dead-letter queues and filtering.

Note: A partner-managed Apache Kafka service, Confluent Cloud (https://www.confluent.io/gcp/) is available. If you're considering a migration from Kafka to Pub/Sub, consult this migration guide (/architecture/migrating-from-kafka-to-pubsub).

Another feature that Pub/Sub adopts from messaging middleware is **per-message parallelism**, rather than partition-based messaging. Pub/Sub "leases" individual messages to subscriber clients, then tracks whether a given message is successfully processed.

By contrast, other horizontally scalable messaging systems use partitions for horizontal scaling. This forces subscribers to process messages in each partition in order and limits the number of concurrent clients to the number of partitions. Per-message processing maximizes the parallelism of subscriber applications, and helps ensure publisher/subscriber independence.

Note: Partition-based parallelism is used by Pub/Sub Lite, but not Pub/Sub.

Compare Service-to-service and service-to-client communication

Pub/Sub is intended for service-to-service communication rather than communication with end-user or IoT clients. Other patterns are better supported by other products:

- **Client-server.** To send messages between a mobile/web app and a service, use products that include Firebase Realtime Database (https://firebase.google.com/docs/database) and Firebase Cloud Messaging (https://firebase.google.com/docs/cloud-messaging).

- **IoT-client-service.** To send messages between an IoT app and a service, use Cloud IoT Core (/iot/docs/concepts/overview)
- **Asynchronous service calls.** Use Cloud Tasks (/tasks/docs/dual-overview)

You can use a combination of these services to build client -> services -> database patterns. For example, see the tutorial Streaming Pub/Sub messages over WebSockets (/architecture/streaming-cloud-pub-sub-messages-over-websockets).

Integrations

Pub/Sub has many integrations with other Google Cloud products to create a fully featured messaging system:

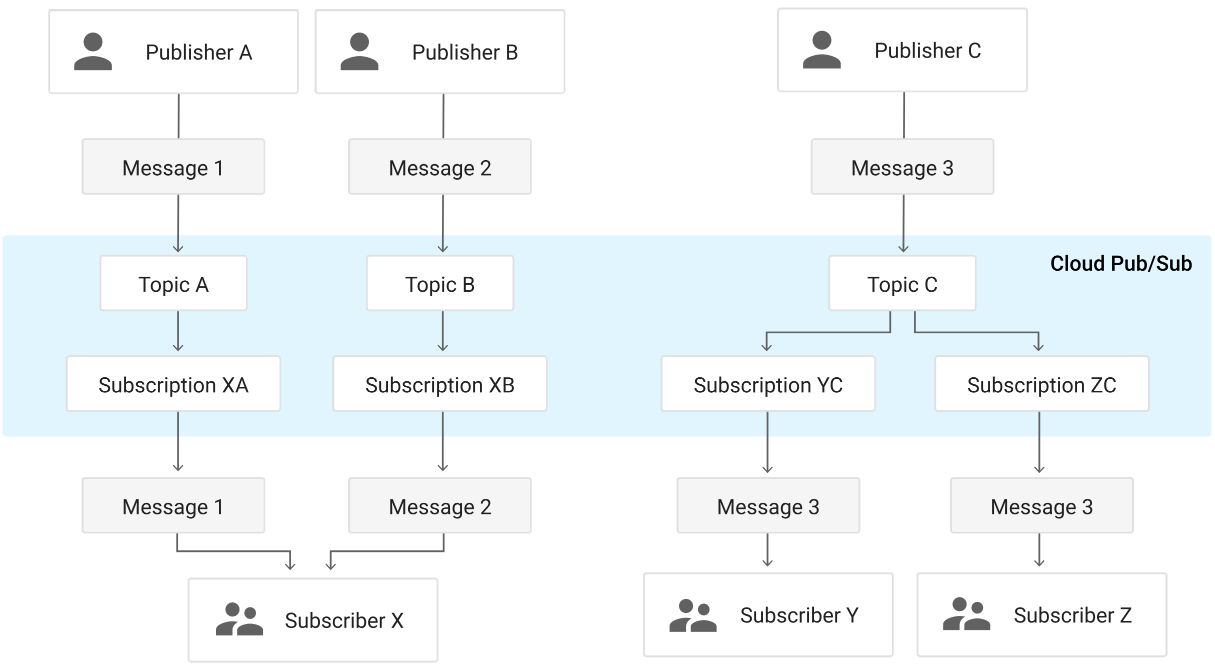
- **Stream processing and data integration.** Supported by Dataflow (/dataflow/docs), including Dataflow templates (/dataflow/docs/concepts/dataflow-templates) and SQL (/dataflow/docs/samples/join-streaming-data-with-sql), which allow processing and data integration into BigQuery and data lakes on Cloud Storage. Dataflow templates for moving data from Pub/Sub to Cloud Storage, BigQuery, and other products are available in the Pub/Sub and Dataflow UIs in the Google Cloud console. Integration with Apache Spark (/learn/what-is-apache-spark), particularly when managed with Dataproc (/dataproc/docs/concepts/overview) is also available. Visual composition of integration and processing pipelines running on Spark + Dataproc can be accomplished with Data Fusion (/data-fusion/docs/concepts/overview).
- **Monitoring, Alerting and Logging.** Supported by Monitoring and Logging products.
- **Authentication and IAM.** Pub/Sub relies on a standard OAuth authentication used by other Google Cloud products and supports granular IAM, enabling access control for individual resources.
- **APIs.** Pub/Sub uses standard gRPC and REST service API technologies (/pubsub/docs/apis) along with client libraries (/pubsub/docs/reference/libraries) for several languages.
- **Triggers, notifications, and webhooks.** Pub/Sub offers push-based delivery of messages as HTTP POST requests to webhooks. You can implement workflow automation using Cloud Functions (/functions/docs) or other serverless products.

- **Orchestration.** Pub/Sub can be integrated into multistep serverless Workflows (/workflows) declaratively. Big data and analytic orchestration often done with Cloud Composer (/composer/docs), which supports Pub/Sub triggers.

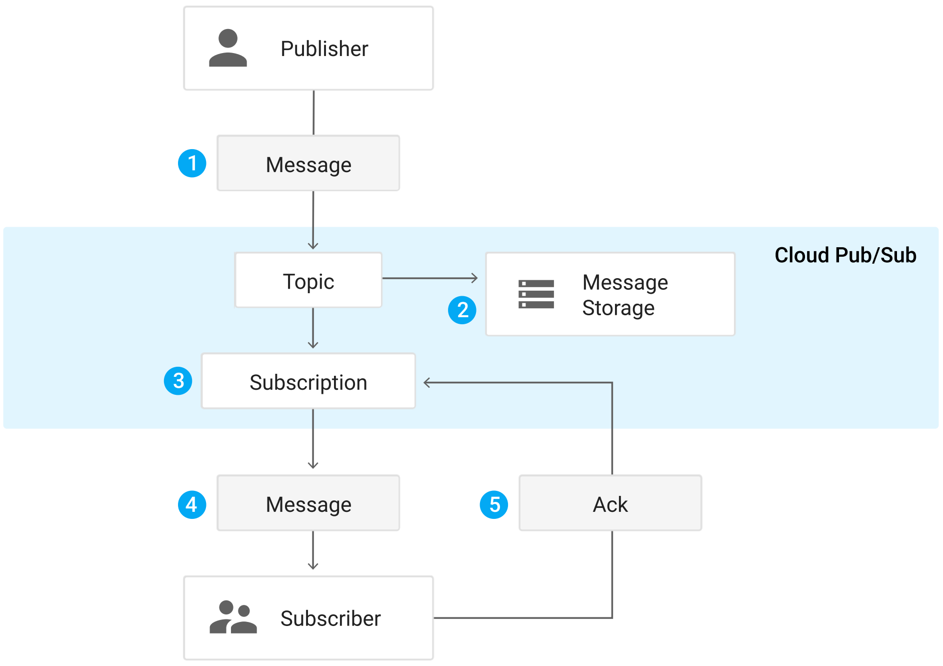
Core concepts

- **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.
- **Publisher.** An application that creates and sends messages to a single or multiple topics.
- **Subscriber.** An application with a subscription to a single or multiple topics to receive messages from it.
- **Acknowledgment (or "ack").** A signal sent by a subscriber to Pub/Sub after it has received a message successfully. Acknowledged messages are removed from the subscription message queue.
- **Push and pull.** The two message delivery methods. A subscriber receives messages either by Pub/Sub **pushing** them to the subscriber chosen endpoint, or by the subscriber **pulling** them from the service.

Publisher-subscriber relationships can be one-to-many (fan-out), many-to-one (fan-in), and many-to-many, as shown in the following diagram:



The following diagram illustrates how a message passes from a publisher to a subscriber. For push delivery, the acknowledgment is implicit in the response to the push request, while for pull delivery it requires a separate RPC.



Next steps

- Get started with the Pub/Sub quickstart (/pubsub/docs/create-topic-console) or the Pub/Sub Lite quickstart (/pubsub/lite/docs/create-topic-console).
- Read the architectural overview (/pubsub/architecture) of Pub/Sub.
- Learn how to build a Pub/Sub messaging system (/pubsub/docs/quickstart-py-mac).
- Understand Pub/Sub pricing (/pubsub/pricing).
- Understand quotas and limits for Pub/Sub (/pubsub/quotas) and Pub/Sub Lite (/pubsub/lite/quotas).
- Read the Pub/Sub release notes (/pubsub/docs/release-notes).
- Explore data engineering with Google Cloud services (https://www.qwiklabs.com/courses/1530?catalog_rank=%7B%22rank%22%3A3%2C%22num_filters%22%3A0%2C%22has_search%22%3Atrue%7D&search_id=10146692) on Qwiklabs.

Last updated 2022-09-30 UTC.