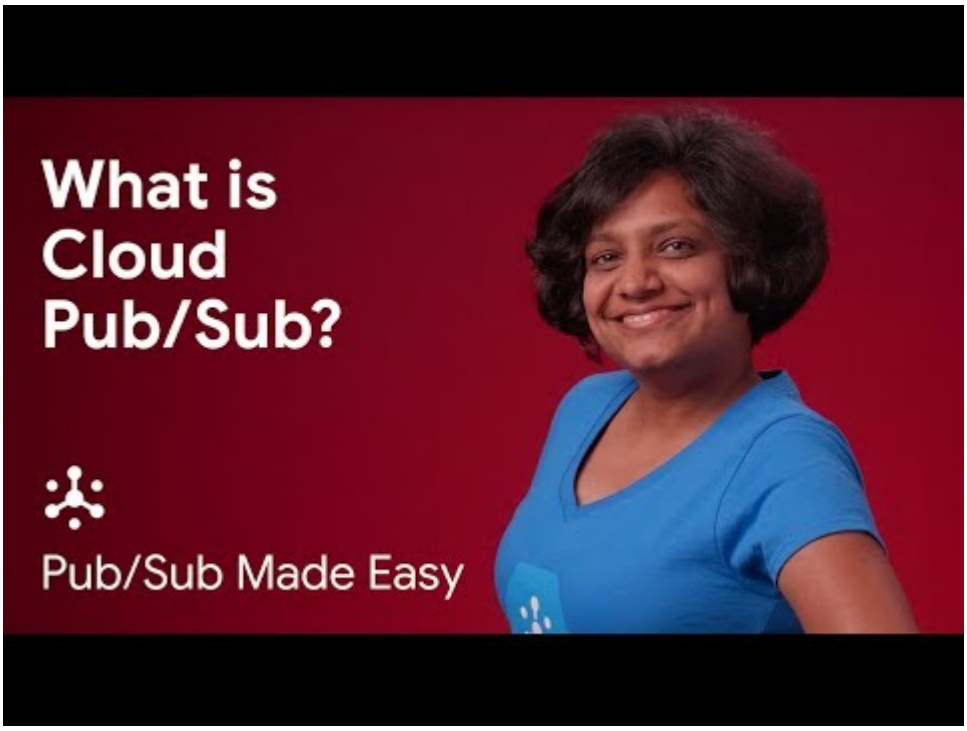
What is Pub/Sub?

cloud.google.com/pubsub/docs/overview



Watch Video At: https://youtu.be/MjEam95VLiI

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 is equally effective as 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 will be processed. Pub/Sub then delivers events to all services that need to react to them. Compared to systems communicating through RPCs, where publishers must wait for subscribers to receive the data, such asynchronous integration increases the flexibility and robustness of the system overall.

To get started with Pub/Sub, check out the Quickstart using Cloud Console. For a more comprehensive introduction, see Building a Pub/Sub messaging system.

Common use cases

- **Ingestion user interaction and server events** To make use of user interaction events from end-user apps or server events from your system, you may forward them to Pub/Sub and then use a stream processing tool such as Dataflow) which delivers them to BigQuery, Bigtable, Cloud Storage and other databases. Pub/Sub allows you to 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. This supports an "enterprise event bus" and event-driven application design patterns. Pub/Sub and allows you to 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 a large number of tasks among multiple workers--such as compressing text files, sending email notifications, evaluating AI models, or reformatting images--by using Pub/Sub messages to connect Cloud Functions.
- **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 IoT devices.** For example, a residential sensor can stream data to backend servers hosted in the cloud.
- **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.

Pub/Sub or Pub/Sub Lite

Pub/Sub consists of two services:

- **Pub/Sub service**: This should be 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 Lite service**: A separate but similar messaging service built for low cost. It offers zonal storage, but requires you to pre-provision and manage storage and throughput capacity.

Consider Pub/Sub Lite only for applications where achieving extremely low cost justifies some additional operational work and lower availability.

For more details, see Choosing Pub/Sub or Pub/Sub Lite. To try Pub/Sub Lite, see Getting started with Pub/Sub Lite.

Comparing Pub/Sub to other messaging technologies

Pub/Sub combines the horizontal scalability of Apache Kafka and Pulsar with features found in traditional messaging middleware like Apache ActiveMQ and RabbitMQ, such as dead-letter queues and filtering.

Note: A partner-managed Apache Kafka service, Confluent Cloud is available on GCP. If you are considering a migration from Kafka to Pub/Sub, consult this migration guide.
Another feature Pub/Sub adopts from messaging middleware is **per-message parallelism** (rather than partition-based). Pub/Sub "leases" individual messages to subscriber clients, then keeps track of whether a given message has been 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 maximises 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.

Service-to-service vs. 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 and Firebase Cloud Messaging.
- **IoT-client-service**: To send messages between an IoT app and a service, use Cloud IoT Core
- **Asynchronous service calls**: Use Cloud Tasks

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.

Integrations

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

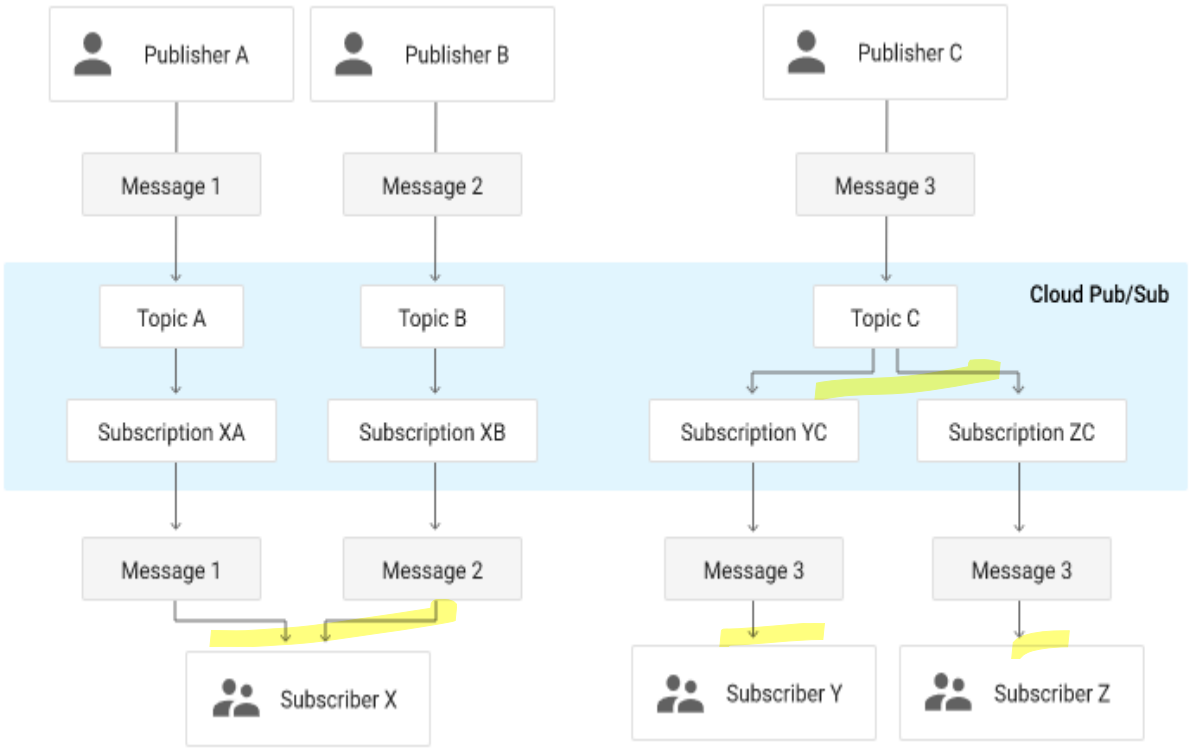
- **Stream processing and data integration:** Supported by Dataflow, including Dataflow templates and SQL, which allow processing and data integration into BigQuery and data lakes on Cloud Storage. Dataflow templates for moving data from Pub/Sub to Cloud Storage, BigQuery and other products are available in the Pub/Sub and Dataflow UIs in the Cloud Console. Integration with Apache Spark, particularly when managed with Dataproc is also available. Visual composition of integration and processing pipelines running on Spark + Dataproc can be accomplished with Datafusion.

- **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 along with client libraries for several languages.
- **Triggers, notifications and webhooks**: Pub/Sub offers push-based delivery of messages as HTTP POST requests to webhooks. This lets you easily implement workflow automation using Cloud Functions 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, which supports Pub/Sub triggers.

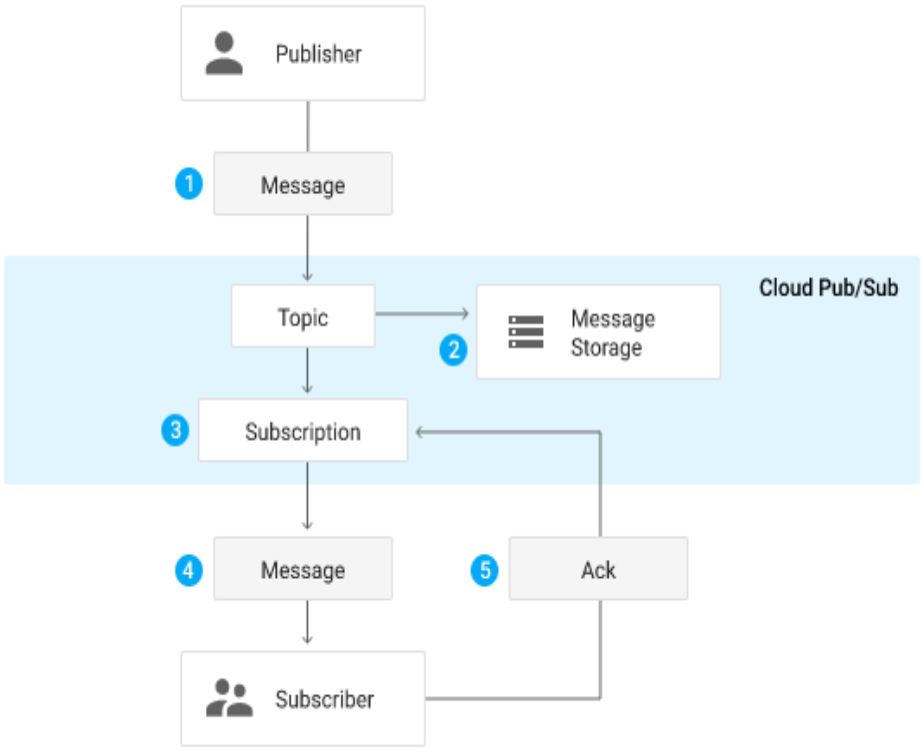
Core concepts

- **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.
- **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 topic(s).
- **Subscriber**: An application with a subscription to a topic(s) to receive messages from it.
- **Acknowledgement (or "ack")**: A signal sent by a subscriber to Pub/Sub after it has received a message successfully. Acked messages are removed from the subscription's message queue.
- **Push and pull**: The two message delivery methods. A subscriber receives messages either by Pub/Sub **pushing** them to the subscriber's 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. Note that for push delivery the ack is implicit in the response to the push request, while for pull delivery it requires a separate RPC.



Next steps

- Get started with the Quickstart using Cloud Console
- Read the Architectural overview to understand the technical implementation Pub/Sub
- Understand pricing
- Explore Data Engineering with Google Cloud services on Qwiklabs

Rate and review