Pub/Sub - Michaël Bettan

Definition: Event-driven asynchronous messaging service that decouples senders (producing events) and receivers (processing events). Allows for secure and highly available communication between independently written applications.	**Availability**: Global (no guarantee of multi-region storage)	**SLA**: >=99.95%

Use Cases:

1. Data streaming from various processes or devices
2. Balancing workloads in network clusters
3. Queue can efficiently distribute tasks
4. Implementing asynchronous workflows
5. Reliability improvement- in case zone failure
6. Distributing event notifications
7. Refreshing distributed caches
8. Logging to multiple systems

Billing:

- **Data throughput**: volume of data processed (GB)
- **Data storage:** Charges apply for storing messages, with different rates for different storage durations.
- **Network usage**: Costs are incurred for data transferred between regions or out of GCP

Core Concepts

- ***Streaming Data:*** *unbounded data (continuous flow of data)*
- ***Messaging service****: Queue (app write > async < app read)*
- ***Loosely coupled architecture****: designing interfaces across modules to reduce the interdependencies across components: Fault tolerance, Scalability, Message Queuing*
- ***Deduplication:*** *action to eliminate duplicate records*
- ***Backlog:*** *accumulation of data to be processed*
- ***Sensors:*** *device emitting events*
- ***Publisher:*** *senders producing events*
- ***Subscriber:*** *receivers processing events*
- ***Dead letter Queue:*** *offline message inspection*
- ***Seek & Replay:*** *ability to reprocess & discard messages*
- ***Ack deadline****: time to wait for ACK by subscriber, push/pull*
- ***Snapshot*** *captures current state at a specific* ***point in time***
- ***Msg Retention Duration*** *after ACK* ***is*** *up to 7 days*

Data Model

- ***Topic, Subscription, Message*** *(combination of data and attributes that a publisher sends to a topic and is eventually delivered to subscribers)*
- ***Message Attribute*** *(key-value pair that a publisher can define for a message)*

Message Flow

1. ***Publisher*** *creates* ***a topic*** *in Pub/Sub and sends* ***messages*** *to the topic.*
2. ***Messages*** *are persisted in a message store until they are delivered and acknowledged by the subscribers.*
3. *Pub/Sub forwards messages from a topic to all of its* ***subscriptions****, individually. Each* ***subscription*** *receives messages either by* ***push*** *or* ***pull*** *methods*
4. ***Subscriber*** *receives pending messages from its subscription and acknowledges messages.*
5. *When a message is* ***acknowledged*** *by the subscriber, it is removed from the* ***subscription's message queue***

Considerations

- *Duplicate & out-of-order delivery is expected*
- *Possible to add message attributes for ordering*
- *At least-once delivery or* ***exactly once delivery***
- *Offers an* ***exactly-once delivery for pull subscriptions****, to distinguish this from* ***exactly-once processing****, which involves additional application-level logic.*
- ***Exactly-once delivery*** *is limited to a* ***single region****. If your subscribers span multiple regions,* ***duplicate deliveries are possible****, even with the feature enabled.*
- *Messages are stored for* ***up to 31 days (7 days default)***
- *Application receives more data than it can process, Data is held until application is ready*
- *Schemas are used to define a standard message structure and they are assigned to topics during creation.*

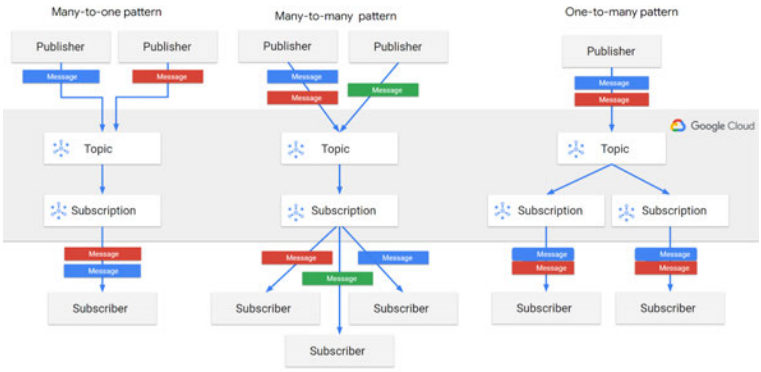
Best Practices

- ***Alerting*** *for backlog thresholds reached*
- *Subscriber(s) must* ***acknowledge the messages*** *(code)*
- *Subscriber(s) should not be* ***under-provisioned*** *(infra scale)*
- *Dataflow solving for out of order messages are resolved*
- *Dataflow solving for* ***at-least-once delivery*** *(duplicates) with an* ***exactly-once processing***
- ***Dead letter Queue*** *and* ***Error logging*** *to prevent failure*
- ***Exponential backoff policy****: add progressively longer delays between retry attempts (for better flow control)*
- *Application endpoint must acknowledge messages within the acknowledgement deadline (otherwise: duplicate/error)*
- ***Message storage policies:*** *Messages are stored in the* ***nearest region by default****. Storage location is configurable via* ***policies*** *(e.g., GDPR). Egress fees may apply.*

Architecture & Message Distribution Patterns

- ***Serverless:*** *No Ops -- no management of servers*
- ***Globally Distributed, Scalable Messaging*** *service that scales effortlessly. Publishers and Subscribers can be located anywhere globally, and the service handles the scaling complexities.*
- ***Pull:*** *Subscribers actively request messages from the subscription. They have to explicitly acknowledge messages after successful processing. This provides more control over the message consumption rate and is suitable for work queues where guaranteed processing is crucial.*
  - *Large volume of messages (more than 1 per sec)*
  - *Critical efficiency and throughput*
  - *No public HTTPS endpoint*
- ***Push:*** *Pub/Sub pushes messages to a registered HTTP endpoint (webhook) on the subscriber. A successful HTTP 200 OK response acts as the acknowledgement. Pub/Sub automatically manages the delivery rate based on the*

subscriber's response times.
    - *Multiple topics for same webbook*
    - *Lower latency*
    - *Dependencies can;t be set up (creds, client libraries)*
- ***One-to-One**: A single publisher sends messages to a single topic, consumed by a single subscriber through a single subscription.*
- ***Fan-in (Load Balancing)**: Multiple publishers send messages to the same topic. Multiple subscribers can consume these messages from the same subscription, enabling parallel processing and load balancing across the subscribers. All subscribers in the subscription receive all messages.*
- ***Fan-out (Distribution)**: A single publisher sends messages to a topic, and multiple subscriptions are attached to that topic. Each subscription receives all the messages, allowing the same data to be processed by different systems.*



Reference: [Overview of the Pub/Sub service]

Security Access Control

- *Project, Topic or Subscription levels*
- *Admin, Editor, Publisher, Subscriber, Viewer roles*

Batching

*Combines multiple messages into a single publish request, optimizing **throughput** and **reducing cost per message**.*

- ***Default behavior**: Enabled by default in client libraries, simplifying implementation.*
- ***Throughput improvement:** Significantly reduces the overhead associated with individual publish requests, allowing for higher message throughput.*
- ***Cost reduction:** Fewer publish requests translate directly to lower costs*
- ***Latency trade-off:** Introduces latency as the publisher waits for enough messages to form a complete batch or a timeout to occur before sending.*
- ***Impact on ordering:** Messages within a batch are guaranteed to be delivered in the order they were added to the batch. However, different batches can arrive at the subscriber out of order. If strict ordering is crucial, consider using a single ordered stream within a topic.*
- ***Flow control and Back pressure:** Batching plays a role in flow control. If the publisher sends messages faster than they can be batched and published, the client library's internal buffers may fill up, leading to backpressure.*

Kafka

- *OSS alternative to Pub/Sub, but without global delivery*
- *Best for hybrid workloads and existing tools/frameworks*
- *Kafka Connect: OSS plugin to connect to GCP world*
- *[Pub/Sub Kafka connector] lets you migrate your Kafka infrastructure to Pub/Sub in phases:*
    - *Sink connector to copy messages to Pub/Sub*
    - *Update individual subscribers with Pub/Sub*
- *Upstream: Source Connector & Downstream: Sink Connector*
- ***Ability to Seek to a Specific Offset:** enables consumers to navigate to a particular offset within a topic, allowing you to read data from any point, including from the very beginning of all captured data.*
- ***Retain Per-Key Ordering:** preserves the sequence of messages within a partition. By associating a key with your messages and routing those messages with the same key to the same partition, you can ensure that the order is maintained for each key.*

Feature	Kafka	Pub/Sub
Persistence	Durable	Ephemeral (options for persistence)
Delivery	At-least-once, at-most-once, exactly-once	At-least-once (exactly-once options)
Ordering	Within a partition	Generally not guaranteed
Scalability	Very high	High
Consumer Mgmt	Consumer-managed offsets	Service-managed
Deployment	Self-hosted or managed service	Managed service
Ecosystem	Rich	Integrates with cloud services

Cloud Logging Metrics

Maintain a healthy subscription

- *Monitor message backlog*
- *Monitor delivery latency health*
- *Monitor acknowledgment deadline expiration*
- *Monitor message throughput*
- *Monitor push subscriptions*
- *Monitor subscriptions with filters*
- *Monitor forwarded undeliverable messages*

Maintain a healthy publisher

- *Monitor message throughput*

Message Rates

- ***subscription/num_undelivered_messages**: high counts indicate potential subscriber issues (processing too slow, crashes, etc.)*
- ***subscription/byte_cost**: track message size to optimize costs and identify large messages that might impact performance.*
- ***topic/send_message_operation_count**: monitor publish rates to detect anomalies like sudden spikes or drops that could indicate publisher problems or unusual traffic.*

Latency

- ***subscription/oldest_unacked_message_age**: high values signify processing delays in subscribers. Crucial for time-sensitive applications.*
- ***topic/publish_latency**: tracks publish latency, helpful for identifying publisher-side bottlenecks.*

Errors

- ***subscription/pull_request_count**: a high volume may indicate inefficient pull configuration.*
- ***topic/send_message_error_count**: publishing errors to identify issues with publisher code or topic availability.*

Resource Usage

- ***subscription/ack_message_operation_count**: compare against published messages to ensure messages are being processed and acknowledged properly. A significant mismatch could indicate message loss or subscriber problems.*