|  |  |
| --- | --- |
|  | Google Cloud Pub Sub |
| Basics/features | <https://cloud.google.com/pubsub/docs/overview>: Decoupled, asynchronous messaging service |
| Motivation |  |
| Use cases | <https://cloud.google.com/pubsub/docs/overview>: Balance workload, async workflow, distribute notifications, refresh distributed cache, log to multiple systems, stream from multiple processes/devices, reliable improvement |
| Flexibility/scalability | <https://cloud.google.com/pubsub/architecture>: Horizontal scalability (add more devices)  <https://stackshare.io/google-cloud-pubsub>: Easy to set up/use |
| Failure/fault tolerance | <https://cloud.google.com/pubsub/docs/samples/pubsub-publish-with-error-handler>: Publish with error handling  <https://cloud.google.com/architecture/framework/reliability>: Redundancy, rollback, traffic limit, recovery, failure detection, incremental changes, coordinate/document emergency response, capacity management  <https://cloud.google.com/pubsub/docs/publisher>: Failed publications are retried automatically. |
| Limitations/bottlenecks | <https://medium.com/google-cloud/google-cloud-pub-sub-ordered-delivery-1e4181f60bc8>: Across a load-balanced set of subscribers, only 1 subscriber can be in a partition at a time. Thus, there is a limit on parallel processing. Possible solution: Put subscriber on topic with more shards but this means more topics to maintain and migration needs to be done more carefully. |
| Message consumption/delivery/ordering | <https://cloud.google.com/pubsub/docs/publisher>: Publisher makes and send messages to a topic with data, ordering key, and metadata.  <https://medium.com/google-cloud/google-cloud-pub-sub-ordered-delivery-1e4181f60bc8>: Subscribers have message ordering property option.  <https://cloud.google.com/pubsub/docs/subscriber>: Subscriber push/pull |
| Storage/retention | <https://cloud.google.com/pubsub/docs/subscriber>: Messages published before a subscription is made won’t be sent to that subscription. Undelivered messages are deleted within 7 days max by default. |
| Data transmission protocol/accuracy | <https://towardsdatascience.com/publisher-subscriber-model-for-apps-and-data-ingestion-flows-b8ba7e85e992>: HTTP |
| Implementation | <https://www.youtube.com/watch?v=f5DOsB7Nlw0>   * Create Google Cloud account and set up features using website UI. * Set up topic, which 1 publisher detects. * Set up 2 subscribers. * Install Python and Google SDK. * Open 3 terminals for 1 publisher and 2 subscribers. * Messages are published and subscribed efficiently with little delay time in between. |
| Real-world scenarios | <https://stackshare.io/google-cloud-pubsub>: 9GAG, PLAID, Samba Tech, PostClick, etc. |
| Qualitative/quantitative comparisons w/ other technologies | Vs. Kakfa/RabbitMQ: <https://stackshare.io/stackups/google-cloud-pubsub-vs-kafka-vs-rabbitmq> |
| Costs | <https://cloud.google.com/pubsub/docs/overview>: Option of Pub/Sub Lite  <https://cloud.google.com/pubsub/quotas>: Quotas and usage |