**Base Experiment (MQ)**

This experiment is an extension of the [Base experiment](https://github.com/CS-6381/FinalProject/blob/main/DesignOfExperiments/Base.md) that is specific to MQ technologies.

**Qualitative Data**

These metrics are inspired by Zhuangwei Kang's presentation he shared with us. Fill in with information you discover about your chosen MQ.

| **Metric** | **Value** |
| --- | --- |
| Centricity (data-centric or message-centric) | Message-centric. <https://cloud.google.com/pubsub/docs/overview>: Decoupled, asynchronous messaging service |
| Connection (machine-to-machine or point-to-point) | <https://www.youtube.com/watch?v=f5DOsB7Nlw0>: Point to point since communicate using multiple terminals on the same computer |
| Underlying Architecture (decentralized or hub-and-spoke) | <https://cloud.google.com/managed-microsoft-ad/docs/hub-spoke>: Hub and spoke. You can make a VPC (Virtual Private Cloud) in Google Cloud and connect to other projects/resources. |
| Protocol | <https://towardsdatascience.com/publisher-subscriber-model-for-apps-and-data-ingestion-flows-b8ba7e85e992>: HTTP |
| Transport(s) | <https://cloud.google.com/load-balancing/docs/internal>: Google Cloud Pub Sub handles TCP and UDP when it comes to load balancing. Messages received from Google Cloud Pub Sub can be sent to an application that utilizes a protocol like TCP or UDP (e. g. a Python message sending/listening application).  <https://towardsdatascience.com/publisher-subscriber-model-for-apps-and-data-ingestion-flows-b8ba7e85e992>: HTTP |
| Data Serialization | <https://docs.fastly.com/en/guides/log-streaming-google-cloud-pubsub>: Data input into Google Cloud Pub Sub must be in JSON format. |
| Supports Queuing | <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. |
| Data Type Representation | <https://cloud.google.com/pubsub/docs/reference/rest/v1/PubsubMessage>: JSON that contains various data types such as string and TimeStamp |
| QoS Parameters | <https://cloud.google.com/iot/docs/how-tos/mqtt-bridge>: You can send messages through MQTT bridge to Cloud IoT core, which supports multiple QOS levels. |
| Supports Dynamic Discovery | <https://cloud.google.com/pubsub/architecture>: Google Cloud Pub/Sub dynamically tweaks publishers and subscribers based on message throughput. |
| Communication Patterns | <https://cloud.google.com/pubsub/docs/subscriber>: Subscriber push/pull  <https://medium.com/google-cloud/google-cloud-pub-sub-ordered-delivery-1e4181f60bc8>: Streaming pull, push/pull |
| Abstraction Layer | <https://cloud.google.com/pubsub/architecture>: For abstraction of implementation details, there exists a service proxy among clients and data senders that helps optimize connection for clients. The control plane assigns the publishers and subscribers to the servers. The data plane handles message transmission among publishers and subscribers. |
| Up-front Complexity | <https://cloud.google.com/pubsub/docs/overview>: Option of Pub/Sub Lite  <https://cloud.google.com/pubsub/quotas>: Quotas and usage  <https://cloud.google.com/pubsub/architecture>: Horizontal scalability (add more devices)  <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. |
| Large Implementations | <https://cloud.google.com/pubsub/quotas#:~:text=Resource%20limits,-Note%3A%20It%20is&text=Retains%20unacknowledged%20messages%20in%20persistent,a%20subscription%2C%20the%20subscription%20expires>.: Google Cloud Pub Sub has certain max limits such as 10000 topics per project, 10 MB per message, 10 KB per schema, etc.  <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.  <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 (<https://cloud.google.com/sdk/docs/install>).  • Open 3 terminals for 1 publisher and 2 subscribers.  • Messages are published and subscribed efficiently with little delay time in between. |