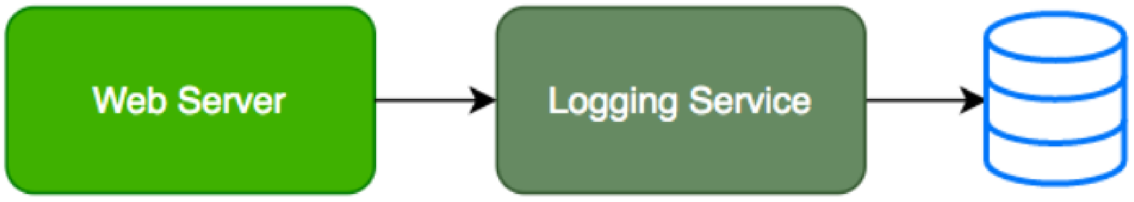
Decoupling Applications with Pub/Sub

Need for Asynchronous Communication

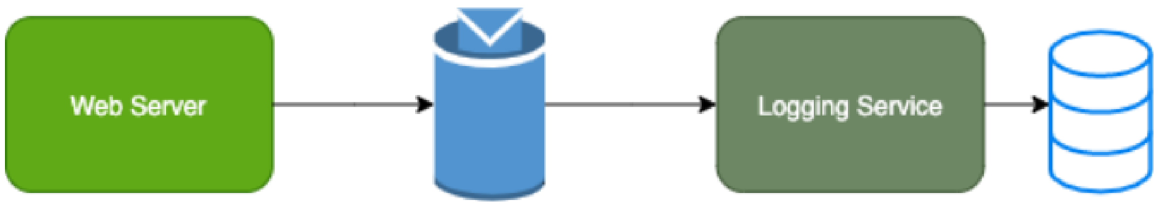
* Why do we need asynchronous communication?

Synchronous Communication



- Applications on your web server make synchronous calls to the logging service
- What if your logging service goes down?
- Will you applications go down too?
- What if all of sudden, there is high load and there are lot of logs coming in?
- Log Service is not able to handle the load and goes down very often.

Asynchronous Communication – Decoupled



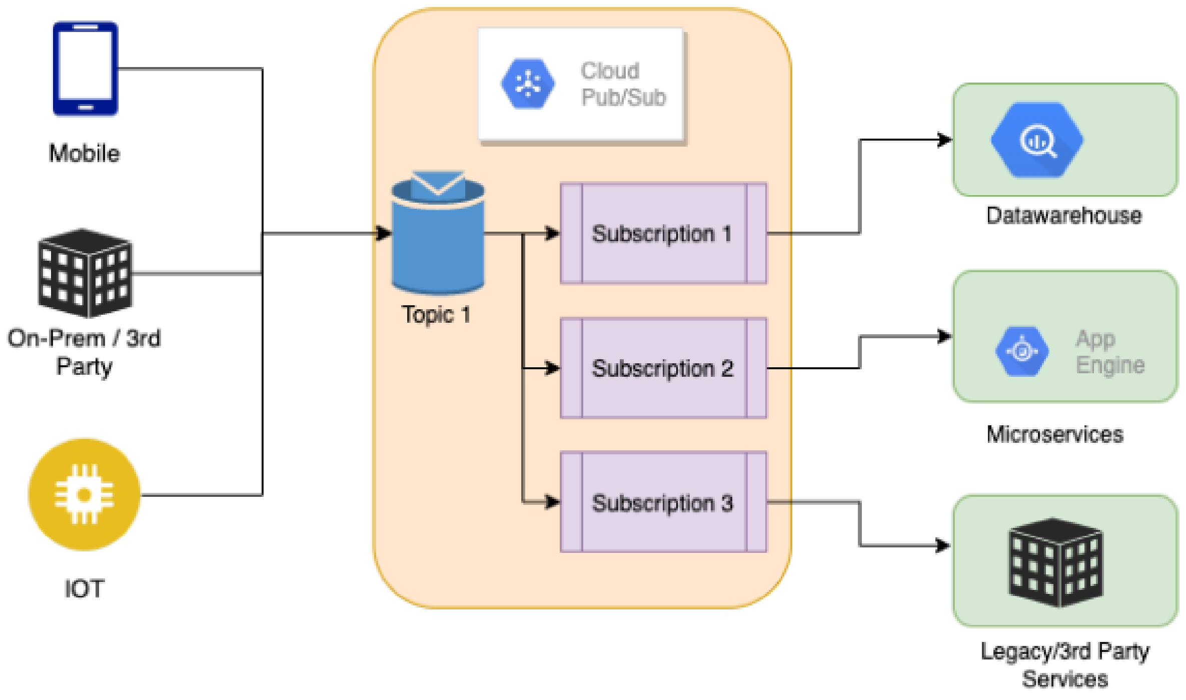
- Create a topic and have your applications put log messages on the topic
- Logging service picks them up for processing when ready
- Advantages:
- Decoupling: Publisher (Apps) don't care about who is listening
- Availability: Publisher (Apps) up even if a subscriber (Logging Service) is down
- Scalability: Scale consumer instances (Logging Service) under high load
- Durability: Message is not lost even if subscriber (Logging Service) is down

Pub/Sub

- Reliable, scalable, fully-managed asynchronous messaging service
- Backbone for Highly Available and Highly Scalable Solutions

- Auto scale to process billions of messages per day
- Low cost (Pay for use)
- Usecases: Event ingestion and delivery for streaming analytics pipelines
- Supports push and pull message deliveries.

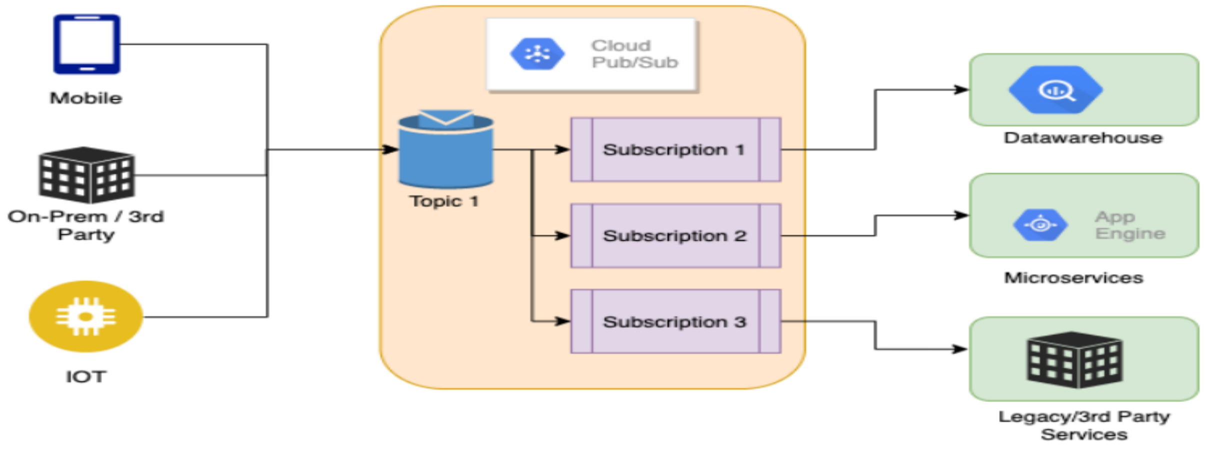
Pub/ Sub - How does it work ?



- Publisher - Sender of a message

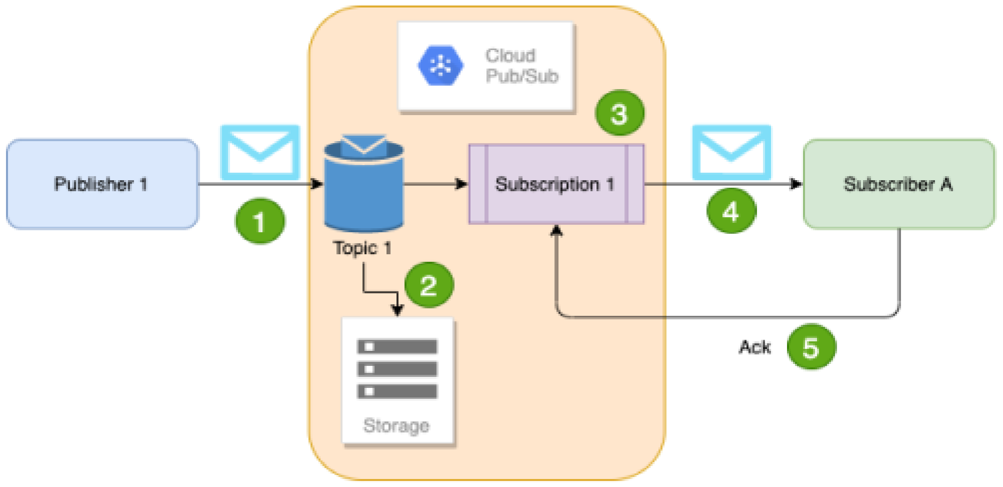
- Publishers send messages by making HTTPS requests to pub sub.google apis.com
- Subscriber - Receiver of the message
- Pull - Subscriber pulls messages when ready
- Subscriber makes HTTPS requests to pubsub.googleapis.c om
- Push - Messages are sent to subscribers
- Subscribers provide a web hook end point at the time of registration
- When a message is received on the topic , A HTTPS POST request is sent to t he web hook endpoints
- Very Flexible Publisher(s) and Subscriber(s) Relationships: One to Many, Many to One, Many to Many

Pub/Sub - Getting Ready with Topic and Subscriptions



- Step 1 : Topic is created
- Step 2 : Subscription(s) are created
- Subscribers register to the topic
- Each Subscription represents discrete pull of messages from a topic:
- Multiple clients pull same subscription => messages split between clients
- Multiple clients create a subscription each => each client will get every message

Pub/Sub - Sending and Receiving a Message



- Publisher sends a message to Topic
- Message individually delivered to each and every subscription
- Subscribers can receive message either by:

- Push: Pub/Sub sends the message to Subscriber
- Pull: Subscribers poll for messages
- Subscribers send acknowledgement(s)
- Message(s) are removed from subscriptions message queue
- Pub/Sub ensures the message is retained per subscription until it is acknowledged

Managing Pub/Sub

- Pub/Sub pubsub
- gcloud pubsub topics create my-topic
- gcloud pubsub subscriptions create my-subscription --topic=my-topic
- --enable-message-ordering - ordered message delivery
- --ack-deadline - how long to wait for acknowledgment?
- --message-filter - criteria to filter messages
- gcloud pubsub subscriptions pull my-subscription --auto-ack --limit
- gcloud pubsub subscriptions ack my-subscription --ack-ids=[ACK_ID,...]
- Topic: gcloud pubsub topics
- gcloud pubsub topics delete my-topic
- gcloud pubsub topics list

- gcloud pubsub topics list-subscriptions my-topic