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

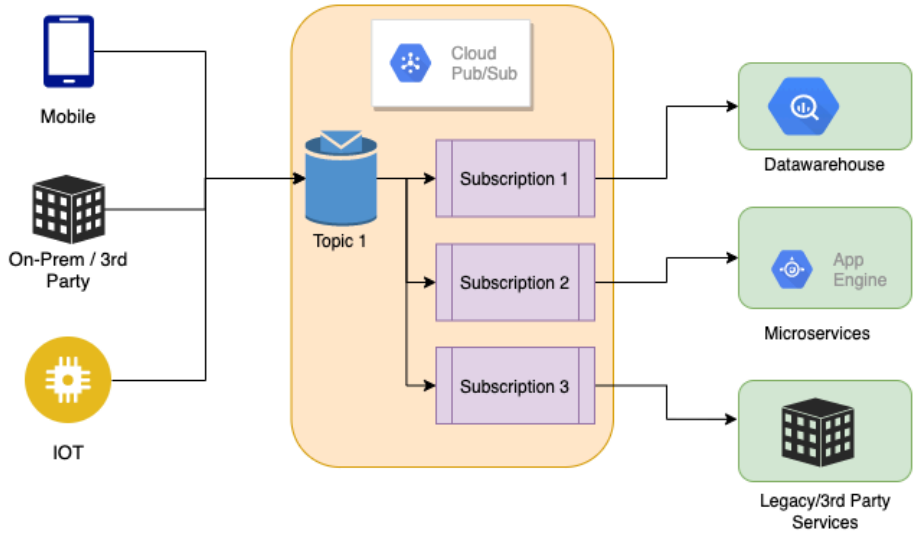
Pub/Sub

In 28 Minutes

- 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

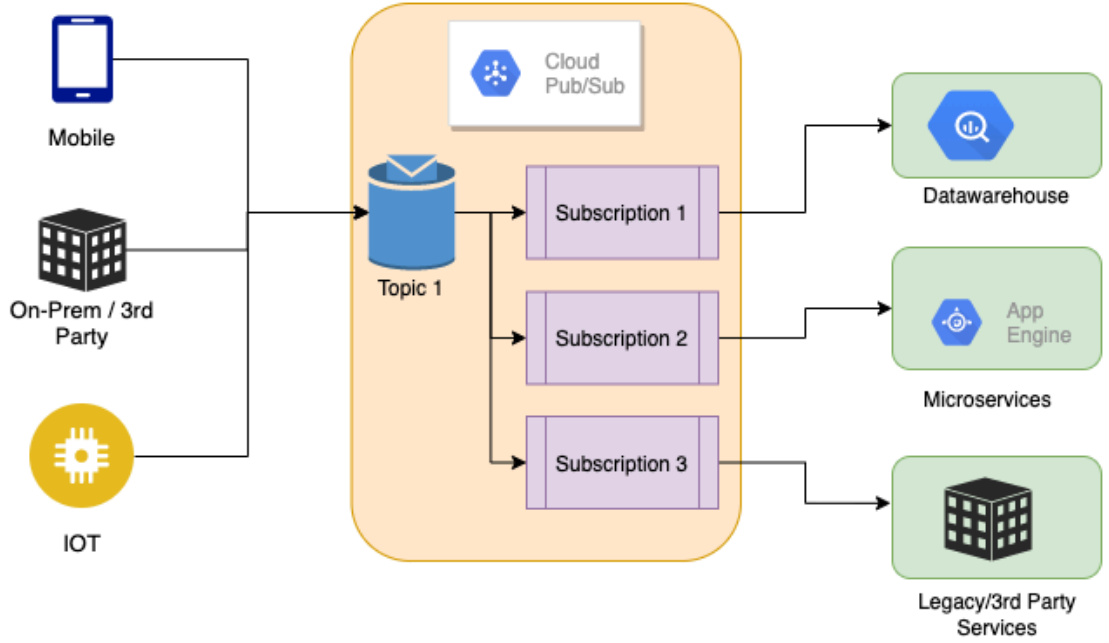
Pub/Sub - How does it work?

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



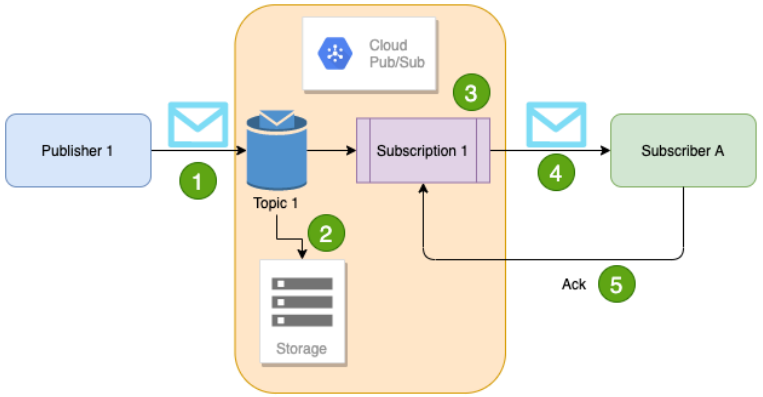
Pub/Sub - Getting Ready with Topic and Subscriptions

- 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

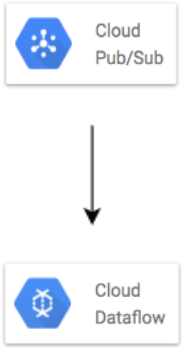
- Publisher sends a message to Topic
- Message **individually** delivered to each and every subscription
  - Subscribers can receive message either by:
    - Push: Pub/Sub sends the message to Subscriber
    - Pull: Subscribers poll for messages
- Subscribers send acknowledgement(s)
- Message(s) are removed from subscriptions message queue
  - Pub/Sub ensures the message is retained **per subscription** until it is acknowledged



Understanding Cloud Pub/Sub Best Practices

In 28 Minutes

- **Usecases**:
  - Convert synchronous to asynchronous workflows
    - Also useful when consumer is unable to keep up with the producer (buffer data)
    - Alternatives: RabbitMQ, Apache Kafka
  - Apply transformations to IOT data stream



- Some use cases need in-order, exactly-once processing (de-duplication) for messages
  - Pub Sub supports in order processing:
    - Option --enable-message-ordering on subscription
  - Add Dataflow into flow to enable message deduplication (exactly-once processing)
    - Maintains a list of message Ids for a time period
    - If a message ID repeats, it is discarded (assumed to be a duplicate)
    - Often sits between data ingestion services (Cloud Pub/Sub, Cloud IOT core ..) and storage/analysis services (Bigtable, BigQuery ..)

Cloud Dataflow

In28 Minutes

- **Cloud Dataflow** is a difficult service to describe:
  - Let's look at a **few example pipelines** you can build:
    - Pub/Sub > Dataflow > BigQuery (Streaming)
    - Pub/Sub > Dataflow > Cloud Storage (Streaming - files)
    - Cloud Storage > Dataflow > Bigtable/CloudSpanner/Datastore/BigQuery (Batch - Load data into databases)
    - Bulk compress files in Cloud Storage (Batch)
    - Convert file formats between Avro, Parquet & csv (Batch)
- **Streaming and Batch Usecases**
  - Realtime Fraud Detection, Sensor Data Processing, Log Data Processing, Batch Processing (Load data, convert formats etc)
- Use **pre-built** templates
- Based on **Apache Beam** (supports Java, Python, Go ...)
- Serverless (and Autoscaling)