



# Design of a Pub-sub System

Dive into designing a pub-sub system and its components.

## We'll cover the following



- First design
- Second design
  - High-level design
- Broker
- Cluster manager
- Consumer manager
- Finalized design
- Conclusion


## First design

In the previous lesson, we discussed that a producer writes into topics, and consumers subscribe to a topic to read messages from that topic. Since new messages are added at the end of the queue, we can use [distributed messaging queues](#) for topics.


The components we'll need have been listed below:

- **Topic queue:** Each topic will be a distributed messaging queue so we can store the messages sent to us from the producer. A producer will write their messages to that queue.
- **Database:** We'll use a relational database that will store the subscription details. For example, we need to store which consumer has subscribed to which topic so we can provide the consumers with their desired messages. We'll use a relational database since our consumer-related data is structured and we want to ensure our data integrity.



- 
- **Message director:** This service will read the message from the topic queue, fetch the consumers from the database, and send the message to the consumer queue.
  - **Consumer queue:** The message from the topic queue will be copied to the consumer's queue so the consumer can read the message. For each consumer, we'll define a separate distributed queue.
  - **Subscriber:** When the consumer requests a subscription to a topic, this service will add an entry into the database.

The consumer will subscribe to a topic, and the system will add the subscriber's details to the database. The producer will write into the topics, and the message director will read the message from the queue, fetch the details to whom it should add the message, and send it to them. The consumers will consume the message from their queue.



**Note:** We'll use fail-over services for the message director and subscriber to guard against failures.

