Google SRE Classroom: Distributed PubSub

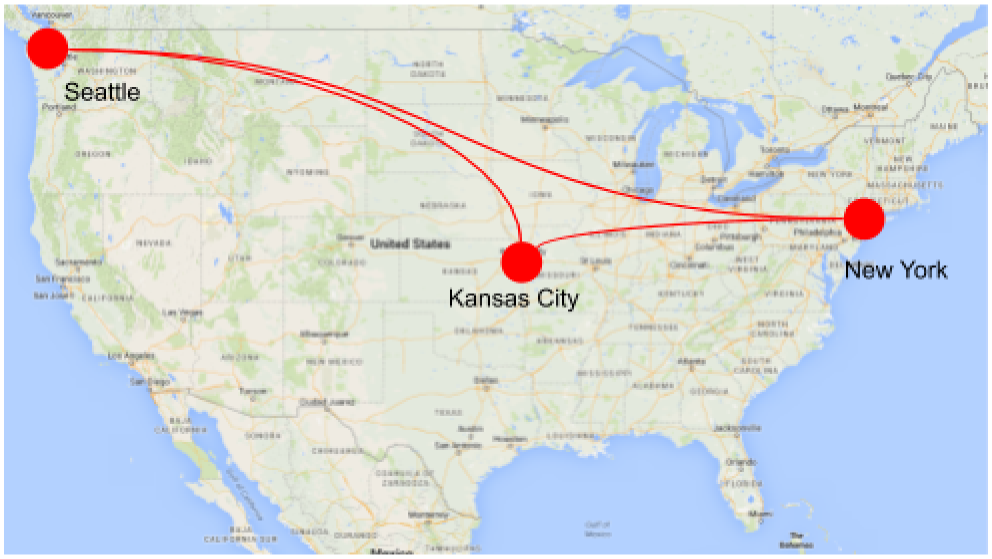
Your Task:

Design a PubSub service that clients all over the world can use to read and write messages.

Producers can publish messages to topics. Consumers consume messages from these topics. Each topic can have multiple producers and consumers. The list of publishers, consumers, and their IDs are known ahead of time.

Messages should be dequeued in the same order that they are enqueued. Each consumer dequeues independently, so the system tracks position per consumer per topic, and on pop requests will pop the next unread message for that consumer and advance the topic marker.

Consumers can begin to consume messages from any topic at any time. They express their intent to start consuming messages from a topic by making a subscribe request. Only messages published after the subscription is created will be delivered to that consumer for that topic.



The system is multi-homed in 3 datacenters: New York, Kansas City and Seattle. If one datacenter goes down for some reason, clients will automatically reconnect to the other two (this is an already-provided feature), and the system must continue functioning. The system must also recover from this failure.

API

- **`subscribe(topic_id, consumer_id)`**
  - Subscribes the given consumer to the given topic.
- **`push(topic_id, message)`**
  - Append a message into the system for the specified topic.
- **`pop(topic_id, consumer_id) -> message`**
  - Read the next message, in order per consumer ID, for the specified topic.
- **`list() -> topics`**
  - List of available topics. *Out of scope*, listed for completeness.

Google SRE Classroom: Distributed PubSub

Traffic

- Topic IDs and consumer IDs are 64 bit integers.
- There are 5,000 topics.
- Each topic has an average of 10,000 messages queued per day, but this is uneven and some topics have much more traffic.
- There are 10,000 consumers. The consumer IDs are pre-shared.
- Each consumer consumes on average consumes 5 topics
- The average message size is 5kB (= 5k bytes), with a maximum size of 100MB.
- 95% of reads occur on data which is 10 minutes old or less.

Required Level of Service (SLO)

- Any two datacenters are able to serve peak load.
- 99% of messages are available for pop'ing from any location in under 1 second.
- 99% of operations succeed in under 500 milliseconds.
    - Faster is better for business though!
- The system can lose up to 0.01% of pushed messages per year.
- At-least once delivery: every non-lost message will be delivered to consumers when requested.
- Messages must be retained for 100 days; older messages can be discarded. These SLOs do not apply to messages older than 100 days.

Completing 99% of operations in 500 ms is an aggressive target. When working with aggressive and possibly-infeasible targets, a good practice is to design the best-performing solution you can, and then try iterating towards a better solution.

Infrastructure

- 3 Datacenters: New York, Kansas City and Seattle
- Datacenters are connected with a 100Gbps connection
- Network availability: 99.99%
- Reliable distributed storage system (1 second replication delay)
- Authentication and Authorization already taken care of;
  all publishers and consumers are already authenticated and trusted clients
- You can use software infrastructure components if you can explain how they work (at a high level), and estimate how they will scale and perform.

Machines available:

- 128GB RAM +
  { 1 x 2TB SSD, or 1 x 4TB HDD }
- 32 cores
- 10 Gbps ethernet connection

Assume the probability of a machine failing some number of times every day is:

# Failures	Probability
0	40%
1	40%
2	15%
3	4%
> 3	1%