**Concept and Coding Video**

https://youtu.be/oVZtzZVe9Dg?si=zj3bGZV2R-tKI9cg

**Distributed Messaging Queue - Notes**

**1. Use Cases**

a. **Asynchronous Processing**

* Common for tasks like sending notifications, emails, logging, etc.
* **Decouples** producer and consumer lifecycles.
* **Reduces latency** for producer: it doesn't block waiting for consumer to process.

b. **Retry Mechanism**

* If a consumer crashes or is temporarily down, the message remains in the queue.
* Supports reprocessing or redelivery attempts.

c. **Rate Mismatch Handling (Pace Matching)**

* Producers can be faster than consumers.
* Queue acts as a buffer to absorb the difference.
* Allows consumers to **process at their own pace**.

**2. Messaging Queue System Components**

a. **Producer**

* Sends (publishes) messages to the queue or topic.

b. **Queue / Topic**

* Stores messages temporarily until a consumer processes them.
* Can be FIFO (first-in, first-out) or support other ordering.

c. **Consumer**

* Retrieves and processes messages from the queue.

**3. Messaging Patterns (Types)**

a. **Point-to-Point (Queue Model)**

* Each message is consumed by **only one** consumer.
* Used for **load balancing**.

b. **Publish-Subscribe (Pub/Sub Model)**

* Messages are broadcast to **multiple subscribers**.
* **Exchange** (in RabbitMQ) or **Topic** (in Kafka) is responsible for distributing messages.

**4. Kafka Messaging Queue Components**

a. **Producer**

* Sends messages to Kafka topics.

b. **Consumer**

* Subscribes to topics and reads messages.

c. **Consumer Group**

* Set of consumers sharing the load of consuming messages from a topic.
* **Each partition is assigned to only one consumer within a group** (no sharing).

d. **Broker**

* Kafka server that stores messages and handles read/write requests.

e. **Cluster**

* Collection of Kafka brokers working together.
* Enables **scalability and fault tolerance**.

f. **Topic**

* A named stream of messages.
* **Logically split into partitions**.

g. **Partition**

* Unit of parallelism.
* **Maintains order** of messages within itself.
* Distributes load across brokers.

h. **Offset**

* Sequential ID assigned to messages in a partition.
* **Committed offset**: the last successfully processed message.

i. **Partition Assignment Rule**

* **A single partition is read by only one consumer in a group**.
* Multiple consumer groups can independently read the same partition.

j. **Zookeeper (or Kafka internal Coordinator in newer versions)**

* **Manages cluster metadata**.
* Handles:
  + **Leader election** (for partitions)
  + **Consumer group coordination**
  + **Offset tracking** (in older Kafka versions; newer uses internal \_\_consumer\_offsets topic)

k. **Fan-in / Fan-out Support**

* Kafka supports **both**:
  + **Fan-in**: multiple producers to same topic.
  + **Fan-out**: multiple consumers read from the same topic via different consumer groups.

l. **Dead Letter Queue (DLQ)**

* Messages that fail after multiple retries can be moved here.
* Helps with debugging and manual intervention.

m. **Message Structure**

* **Key** (optional): used for partitioning logic.
* **Value**: the actual message.
* **Topic** and **Partition**: metadata for routing and storage.

n. **Replication**

* **Partitioning**: splits topic across partitions for parallelism.
* **Replication**: each partition has one leader and multiple follower replicas.
  + **Only the leader** handles read/write.
  + Followers **sync data from leader**.

**5. Kafka Common Interview Questions**

1. **What happens when queue size is exceeded?**
   * Backpressure can be applied;
   * Data can be persisted via scaling (more brokers/partitions);
   * Or oldest messages may be dropped based on retention policy.
2. **What if a broker goes down?**
   * A **follower replica becomes leader** automatically.
   * Ensures high availability.
3. **What if a consumer fails?**
   * Another consumer in the **same group** takes over based on committed offset.
4. **What if a consumer can’t process a message?**
   * Retry a configurable number of times.
   * Then move to **Dead Letter Queue**.
5. **How is Kafka distributed?**
   * Kafka is distributed via **multiple brokers**, each handling multiple partitions.

**6. RabbitMQ Components**

a. **Push Model**

* Messages are **pushed to consumers** (vs. Kafka’s pull model).
* Better for real-time, low-latency use cases.

b. **Exchange**

* Accepts messages from producers, routes them to queues using routing keys.
* Types:
  + **Direct Exchange**: exact match of routing key.
  + **Topic Exchange**: wildcard match (e.g., user.\*, order.#).
  + **Fanout Exchange**: broadcasts to all bound queues.
  + **Headers Exchange**: routing based on headers instead of keys.

c. **Queue**

* Holds messages until consumed.
* Multiple consumers can subscribe to a queue, but **each message is delivered to only one**.

A diagram of a computer server

AI-generated content may be incorrect.

https://notebook.zohopublic.in/public/notes/u3i1s522a981ed32d48bcbb0b940ee3d58f22