
Kafka Day 7 – Revision Notes (Exactly-Once Semantics & Reliability)

---

1. Kafka Partition Ordering Guarantee

- Kafka preserves order **within a partition**.
- If a producer fails mid-transaction, Kafka **aborts** the transaction.
- All writes in that transaction are discarded (atomicity guaranteed).

---

2. Backpressure Handling in Consumers

- Use `max.poll.records` to limit records fetched per poll.
- Slow downstream? Buffer internally or use async handoff.
- Monitor lag & alert early to prevent snowballing delays.

---

3. Exactly-Once Processing Semantics

- Use **idempotent producers** + **transactional producers** + **atomic offset commit**.
- Commit offsets only **after** DB/sink confirms success.
- Guarantees messages aren't duplicated or lost.

---

4. Consumer Lag Monitoring

- Lag = `latest offset - committed offset`.
- Tools: **Burrow**, **Prometheus**, **Grafana**, **Kafka Manager**.
- Watch out for spikes during rebalancing or processing delays.

---

5. Ensuring Durability and Availability

- Use:
    - `replication.factor = 3`
    - `min.insync.replicas = 2`
    - `acks = all`
- Disable unclean leader election.
- Use rack-awareness for better fault tolerance.

---

6. Schema Evolution in Kafka

- Use **Avro**/**Protobuf** + **Schema Registry**.
- Compatibility types:
    - BACKWARD (old consumers work with new schema)
    - FORWARD (new consumers work with old schema)
    - FULL (both ways).
- Always handle defaults for missing fields.

---

7. Real-Time + Batch Design with Kafka

- Use **separate consumers** for real-time (e.g. Spark Streaming) and batch (e.g. Flink windowed jobs).
- Use `log.retention.hours` or compacted topics to allow late replays.

---

8. Kafka Monitoring & Alerting

- Monitor:

    - Under-replicated partitions
    - Consumer lag
    - Broker disk usage
    - Request throughput
- Use: **JMX**, **Prometheus**, **Grafana**, **Datadog**, **Burrow**.

---

9. Ordering Across Partitions

- Kafka **does not guarantee** ordering across partitions.
- You must design using:
    - Single partition per key
    - Sink-side ordering with buffering (careful of memory pressure).

---

10. Kafka Cluster Migration with Minimal Downtime

- Use **MirrorMaker2** to replicate topics across clusters.
- Validate data parity via topic offsets or hash checksums.
- Gradually shift consumers and producers.