Great question — thinking ahead about **failure scenarios** is what separates solid real-time systems from fragile ones. Let’s break down **real-world failure cases** that you need to handle at different stages of a Kafka consumer pipeline like yours.

**✅ Real-World Failure Scenarios in Kafka + Redis Message Processing**

**🔹 1. Consumer Crashes Immediately After Kafka Polls**

**Scenario:**  
Kafka delivers a message to the consumer (@KafkaListener), but the app crashes before processing it or writing to Redis.

**Impact:**

* Kafka may consider the message **consumed** (if auto-commit is enabled)
* Message is lost unless retries or manual commits are used

**How to Handle:**

* ❌ Disable auto-commit (enable-auto-commit: false)
* ✅ Use **manual commit** after writing to Redis
* ✅ Use AckMode.MANUAL\_IMMEDIATE to commit only after safe storage

**🔹 2. Partial Writes to Redis (Redis crash or timeout)**

**Scenario:** Message is parsed, but Redis is unavailable or fails to store the data.

**Impact:**

* Message stays unprocessed
* Next messages (e.g. Footer) may arrive before required OrderItems
* Can lead to permanent hanging groups

**How to Handle:**

* ✅ Wrap Redis writes in retry-safe operations
* ✅ Track Redis failures in a dead-letter store or local file
* ✅ Use RedisTemplate.executeWithStickyConnection() if needed

**🔹 3. Consumer Crashes Mid-Aggregation**

**Scenario:**  
All messages are read and partially written to Redis, but app crashes before Footer is written or group is processed.

**Impact:**

* Redis contains incomplete group data
* Group may never be processed if Footer is never written again

**How to Handle:**

* ✅ Implement **TTL expiration** in Redis per group
* ✅ Use a **rescue job** (scheduled) to re-scan and process stale but complete groups

**🔹 4. Duplicates (Producer retries, network blips)**

**Scenario:**  
Producer resends the same message due to retry, or the consumer reprocesses it due to rebalancing.

**Impact:**

* Duplicate writes in Redis (e.g., same OrderItem stored multiple times)

**How to Handle:**

* ✅ De-duplicate messages in Redis by using unique IDs or sorted sets
* ✅ Consider Redis hash keys like orderItem:<correlationId>:<orderItemId>

**🔹 5. Out-of-Order Message Delivery**

**Scenario:** Header arrives late or after Footer due to producer thread logic or Kafka partitioning.

**Impact:**

* Consumer may not be able to complete the group in a timely fashion

**How to Handle:**

* ✅ Ensure messages with same correlationId go to same Kafka partition (you’ve already done this!)
* ✅ Use Redis TTL to hold incomplete groups
* ✅ Schedule recovery logic to clean or retry stalled groups

**🔹 6. Redis Data Loss (eviction, restart, no persistence)**

**Scenario:**  
Redis restarts or evicts keys due to memory pressure (especially with no AOF or RDB snapshot)

**Impact:**

* Aggregated groups are lost
* Consumer cannot resume from failure state

**How to Handle:**

* ✅ Enable Redis AOF or snapshotting
* ✅ Backup groups to disk or external store
* ✅ Or persist groups to DB after completion

**🔹 7. Listener Exceptions Not Propagated**

**Scenario:**  
Exception is thrown while processing message, but KafkaListener doesn’t retry or dead-letter it.

**Impact:**

* Message is lost silently
* Consumer group offset may move forward

**How to Handle:**

* ✅ Configure a proper SeekToCurrentErrorHandler or DLQ handler
* ✅ Optionally re-queue failed messages back to Kafka or DLQ topic

**🧠 Next Steps**

Here’s how you can tackle these next:

| **Goal** | **Strategy** |
| --- | --- |
| Ensure Redis write before Kafka offset commit | 🔁 Manual acknowledgment mode |
| Handle Redis failures | 🔧 Retry + DLQ on critical |
| Ensure recoverability after crash | 🧰 TTL + scheduled rescans |
| Handle out-of-order & missing messages | 🧠 Smart correlation tracking |
| Ensure message durability | 💾 Redis persistence (AOF) |
| Graceful retry / DLQ | 🛡️ ErrorHandler or DLQ topic |

Would you like me to help you implement:

1. ✅ Manual offset commits after Redis write
2. ✅ Error handler with retries / DLQ
3. ✅ Redis TTL per group
4. ✅ Background scanner to detect orphaned complete groups?

Let’s build it step-by-step 🚀