Apache Kafka Troubleshooting Runbook Pack

A senior engineer's incident runbook for consumer lag, rebalance storms, under-replicated partitions, and broker failures.
DevOps AI ToolKit · devopsaitoolkit.com

Kafka fails in a handful of recognizable shapes: consumers fall behind, the group rebalances in a loop, partitions lose replicas, a broker drops out, producers time out, or the metadata layer (KRaft controllers or ZooKeeper) loses quorum. The trap is treating a symptom on one broker as the root cause when the real problem is replication, the controller, or the client config. Work outside-in: confirm the cluster and controller are healthy first, then localize to a topic, partition, group, or single broker before you touch anything.

> Examples use `broker-1:9092` and topic `orders`. Point `--bootstrap-server` at any live broker. Run every read-only `--describe` first; only change configs or restart with a change record and, ideally, a second engineer. Never enable unclean leader election to "fix" an outage without understanding the data-loss trade-off.

1. First-Response Triage

```
# Is the cluster reachable and which API versions does each broker speak?
kafka-broker-api-versions.sh --bootstrap-server broker-1:9092 | head

# Cluster + controller + broker inventory (KRaft or ZK)
kafka-metadata-quorum.sh --bootstrap-server broker-1:9092 describe --status 2>/dev/null \
  || kafka-cluster.sh cluster-id --bootstrap-server broker-1:9092

# Full picture of the topic in question
kafka-topics.sh --bootstrap-server broker-1:9092 --describe --topic orders
```

- `--describe` on a topic shows `Leader`, `Replicas`, and `Isr` per partition. Note any `Leader: none` or `Isr` shorter than `Replicas` — that is your incident.
- If `kafka-broker-api-versions.sh` hangs or times out for one broker, that broker is the suspect; if it fails for all, it is a bootstrap/DNS/listener or metadata-quorum problem, not a topic problem.
- Establish blast radius: one partition, one topic, one broker, or the whole cluster.

2. Consumer Lag

```
# Lag per partition for a group — the single most useful command
kafka-consumer-groups.sh --bootstrap-server broker-1:9092 \
  --describe --group orders-consumer

# Is the group actually assigned and consuming (state + members)?
kafka-consumer-groups.sh --bootstrap-server broker-1:9092 \
  --describe --group orders-consumer --state --members
```

- The `LAG` column is `LOG-END-OFFSET` minus `CURRENT-OFFSET`. Steady lag = throughput mismatch; lag climbing on **one** partition = a hot key or a stuck consumer on that partition.
- `CONSUMER-ID` / `HOST` blank with lag present → no member owns that partition (group is down, paused, or mid-rebalance). Go to §3.
- State `Stable` with growing lag → consumers are too slow: add members (up to the partition count), speed up processing, or check for a poison message blocking `poll()`.
- Do **not** reset offsets to hide lag. If you must skip, do it deliberately: `--reset-offsets --to-latest --group orders-consumer --topic orders --execute` (only after `--dry-run`).

3. Rebalance Storms

```
# Watch for repeated (Re-)Join / rebalance churn in the group coordinator's log
grep -E "Rebalance|Preparing to rebalance|member.*(join|leav)" \
  /var/log/kafka/server.log | tail -40

# Client-side smoking gun in the consumer application log
grep -E "CommitFailedException|rebalance|max.poll.interval" app.log | tail
```

- `CommitFailedException` means the consumer took longer than `max.poll.interval.ms` (default 300000) between `poll()` calls, so the coordinator evicted it and reassigned its partitions. Fix the slow processing or raise `max.poll.interval.ms` — not `session.timeout.ms`.
- `session.timeout.ms` (default 45000) governs heartbeat liveness, bounded by broker `group.min/max.session.timeout.ms`. Aggressive GC or network blips trip it and trigger churn.
- Members constantly leaving/rejoining with the same IDs → enable **static membership**: set `group.instance.id` on each consumer so a restart rejoins in place instead of forcing a full rebalance.
- Prefer the cooperative-sticky assignor (`partition.assignment.strategy`) to avoid stop-the-world reassignments during scaling.

4. Under-Replicated & Offline Partitions

```
# Partitions missing replicas from the ISR (should return NOTHING when healthy)
kafka-topics.sh --bootstrap-server broker-1:9092 --describe --under-replicated-partitions

# Partitions with no leader at all — data unavailable
kafka-topics.sh --bootstrap-server broker-1:9092 --describe --unavailable-partitions

# What durability is actually required for writes?
kafka-configs.sh --bootstrap-server broker-1:9092 --describe \
  --entity-type topics --entity-name orders | grep -E "min.insync.replicas|replicas"
```

- `Isr` shorter than `Replicas` means a follower stopped fetching — usually that broker is down, slow, or its disk is full. Find the missing broker ID and check §5/§8.
- If in-sync replicas drop **below** `min.insync.replicas` (commonly 2 with RF 3), producers using `acks=all` get `NotEnoughReplicasException` and writes stop. That is by design — it protects durability. Restore a replica; do not lower `min.insync.replicas` mid-incident.
- Offline / unavailable partitions with `Leader: none` need a live in-sync replica to become leader, or a deliberate recovery decision (see §5).

5. Leader Election & Broker Failures

```
# Who is the active controller? (KRaft)
kafka-metadata-quorum.sh --bootstrap-server broker-1:9092 describe --status \
  | grep -E "LeaderId|CurrentVoters"

# Client errors that indicate stale metadata after a leader move
grep -E "NotLeaderForPartition|LeaderNotAvailable|UnknownTopicOrPartition" app.log | tail
```

- `NotLeaderForPartition` / `LeaderNotAvailable` right after a broker restart are usually transient — clients refresh metadata and recover within a few seconds. Persistent ones mean no replica can take leadership (all out of ISR).
- When a partition's ISR is empty and its leader is gone, Kafka will **not** auto-elect by default (`unclean.leader.election.enable=false`), trading availability for zero data loss. Enabling unclean election (`kafka-leader-election.sh --election-type UNCLEAN` or the config) can bring the partition back **but may lose committed messages**. This is a conscious, logged decision — never a reflex.
- Prefer a clean `PREFERRED` election after the real leader returns:

  ```
  kafka-leader-election.sh --bootstrap-server broker-1:9092 \
    --election-type PREFERRED --all-topic-partitions
  ```

6. Producer Errors

```
# Confirm broker-side message-size ceilings before blaming the producer
kafka-configs.sh --bootstrap-server broker-1:9092 --describe \
  --entity-type topics --entity-name orders | grep -E "max.message.bytes|retention"
grep -E "RecordTooLargeException|TimeoutException|ProducerFenced|NotEnoughReplicas" app.log | tail
```

- `RecordTooLargeException` → the record exceeds producer `max.request.size` (default ~1 MB) or topic `max.message.bytes`. Raise both consistently, or compress/split the payload — a batch can also exceed the limit even when single records fit.
- `TimeoutException` on send → `delivery.timeout.ms` elapsed. Causes: partition has no leader (§5), ISR below `min.insync.replicas` (§4) with `acks=all`, or the broker is overloaded (§9). Increasing `retries` alone just delays the failure.
- Set `acks=all` + `enable.idempotence=true` for durability and no duplicates; `acks=1` risks loss on leader failover.
- `ProducerFencedException` → a newer producer instance with the same `transactional.id` took over (zombie fencing). Expected during failover; only alarming if it loops, which points to two live instances sharing an id.

7. KRaft / ZooKeeper Quorum

```
# KRaft: controller quorum health, leader, lag of each voter
kafka-metadata-quorum.sh --bootstrap-server broker-1:9092 describe --status
kafka-metadata-quorum.sh --bootstrap-server broker-1:9092 describe --replication

# ZooKeeper mode (pre-KRaft clusters): is each node ok and who is leader?
echo ruok | nc zk-1 2181 ; echo srvr | nc zk-1 2181 | grep -E "Mode|Zxid"
```

- KRaft: a healthy quorum shows one `LeaderId`, all `CurrentVoters` present, and small/zero `Lag` per follower. Growing follower `Lag` means a controller is falling behind — check its disk and network before it drops from the quorum.
- Quorum needs a majority: 2 of 3 or 3 of 5 controllers. Lose the majority and metadata writes stall — topic creation, leader elections, and config changes all freeze even though data reads may continue.
- ZooKeeper: exactly one `Mode: leader`, the rest `follower`. `imok` from `ruok`. A node that won't join usually has a clock skew, `myid` mismatch, or a full snapshot dir.

- Never mix modes: a cluster is either KRaft or ZooKeeper. Migration is a planned procedure, not an incident action.

8. Disk & Retention

```
# Are any log directories marked offline/failed on a broker?
grep -E "log directory|Failed to (write|read)|offline" /var/log/kafka/server.log | tail
# Effective retention for the topic
kafka-configs.sh --bootstrap-server broker-1:9092 --describe \
  --entity-type topics --entity-name orders | grep -E "retention.ms|retention.bytes"
df -h /var/lib/kafka   # the data mount for log.dirs
```

- A failed entry in `log.dirs` takes every partition on that directory offline; those partitions go under-replicated (§4) and their leaders move to other brokers. Replace/repair the disk, then let the broker re-sync.
- If `df` shows the data mount full, Kafka can crash or refuse writes. Do **not** `rm` segment files by hand — reduce `retention.ms` or `retention.bytes` on high-volume topics and let Kafka delete cleanly, or add disk.
- `cleanup.policy=compact` topics (e.g. `__consumer_offsets`) grow if compaction stalls; confirm log-cleaner threads are alive in the log.

9. Latency & Throughput

```
# Broker JMX metrics that localize a slowdown (via jmxterm or your metrics stack):
#   kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent
#   kafka.network:type=RequestChannel,name=RequestQueueSize
#   kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Produce|Fetch
grep -E "expired|throttle|purgatory" /var/log/kafka/server.log | tail
```

- `RequestHandlerAvgIdlePercent` near 0 → request handler threads are saturated; raise `num.io.threads` (I/O) and check `num.network.threads`. Idle near 1.0 → the bottleneck is elsewhere (disk or downstream).
- A large **Produce/Fetch purgatory** size is normal for `acks=all` and long-poll fetches (requests waiting on replication or data); a sudden spike alongside rising latency points to slow followers (§4) or slow disk.
- Kafka relies on the OS **page cache** for read throughput. Falling page-cache hit rates (rising disk reads) mean consumers are reading old data or memory pressure is evicting the cache — check for a backfilling consumer.
- Confirm the disk itself: high `await`/utilization on the `log.dirs` device caps everything.

10. Security (SASL / TLS / ACLs)

```
# Handshake / auth failures show up broker-side
grep -E "SSLHandshakeException|Authentication failed|SASL|failed authentication" \
  /var/log/kafka/server.log | tail

# What ACLs exist for the principal and topic?
kafka-acls.sh --bootstrap-server broker-1:9092 --list --topic orders

# Verify the broker keystore/truststore contents and expiry
keytool -list -v -keystore /etc/kafka/ssl/kafka.server.keystore.jks | grep -E "Alias|until"
```

- `SSLHandshakeException` → cert/CA mismatch or an **expired** certificate (check the `until` date), or the client doesn't trust the broker CA. TLS failures block the connection before any Kafka error surfaces.
- SASL "Authentication failed" → wrong mechanism (`SCRAM-SHA-512` vs `PLAIN` vs `GSSAPI`), bad credentials, or a JAAS/keytab problem. Confirm client and broker agree on `security.protocol` (`SASL_SSL` etc.).
- `TopicAuthorizationException` / `GroupAuthorizationException` means auth **succeeded** but the principal lacks an ACL — grant the specific `--producer`/`--consumer` right, never a blanket allow-all.

11. Decision Tree

```
Kafka incident: what's the top symptom?
 ├─ Cluster/metadata unreachable? ── yes → §7 quorum: KRaft voters or ZK majority; fix controllers FIRST
 └─ no
    ├─ under-replicated / offline partitions? ── yes → §4/§5: find missing broker; check disk §8; NO unclean election reflex
    ├─ consumers behind?
    │   ├─ lag steady/rising, group Stable? ── §2: scale/ speed consumers, hot partition?
    │   └─ group churning (CommitFailed)?  ── §3: raise max.poll.interval.ms, static membership
    ├─ producers failing?
    │   ├─ RecordTooLarge? ── §6: max.message.bytes / max.request.size
    │   └─ Timeout / NotEnoughReplicas? ── §4 (ISR) + §5 (leader) + §9 (load)
    ├─ slow but working? ── §9: handler idle %, purgatory, page cache, disk await
    └─ connection refused / auth errors? ── §10: TLS expiry, SASL mechanism, ACLs
```

12. Copy/Paste One-Shot Triage

```
BS=broker-1:9092 ; T=orders ; G=orders-consumer
echo "== brokers/api ==="       ; kafka-broker-api-versions.sh --bootstrap-server $BS 2>&1 | grep -c "id:"
echo "== controller quorum ==" ; kafka-metadata-quorum.sh --bootstrap-server $BS describe --status 2>/dev/null | grep -E "LeaderId|CurrentVoters"
echo "== under-replicated ==="  ; kafka-topics.sh --bootstrap-server $BS --describe --under-replicated-partitions
echo "== offline ==="           ; kafka-topics.sh --bootstrap-server $BS --describe --unavailable-partitions
echo "== topic ISR ==="         ; kafka-topics.sh --bootstrap-server $BS --describe --topic $T
echo "== group lag ==="         ; kafka-consumer-groups.sh --bootstrap-server $BS --describe --group $G
echo "== group state ==="       ; kafka-consumer-groups.sh --bootstrap-server $BS --describe --group $G --state
```

13. Incident Notes Template

```
INCIDENT: Apache Kafka
Started (UTC):        ____    Detected by: ____    Sev: ____
Impact:               (topic(s) / group(s) / producers / % of traffic)
Blast radius:         [ ] one partition [ ] one topic [ ] one broker [ ] whole cluster
Metadata layer:       [ ] KRaft quorum OK: voters __/__  leader ____
                      [ ] ZooKeeper OK: leader ____ nodes __/__
Replication:          under-replicated: ____  offline: ____  min.insync.replicas: ____
Consumers:            group ____  lag ____  state ____  rebalancing? Y/N
Producers:            error ____ (RecordTooLarge / Timeout / NotEnoughReplicas / Fenced)
Broker/disk:          down broker id ____  log.dirs full? ____  offline log dir? ____
Action taken:         (config changed / restart / leader election + who + when)
Unclean election?     [ ] no  [ ] YES — data loss accepted, approved by ____
Result:               [ ] recovered @ ____   [ ] escalated to ____
Root cause:           ____
Follow-ups:           (ISR alerts, retention, worker scaling, cert expiry) ____
```