

08 The 10 IBM MQ Reason Codes I See Most in Production

Decode any code with `mqrc <code>` on an MQ host. In Java/JMS stacks the code shows up inside messages such as `JMSWMQ2007 / JMSCMQ0001`, followed by "reason 2035 ('MQRC_NOT_AUTHORIZED')". For connection and security problems, always read the **queue manager's** `AMQERR01.LOG` as well. The client only gets the code; the explanation is in the server log.

RC	Constant	Typical causes	Fix / next step
2035	MQRC_NOT_AUTHORIZED	Blocked by CHLAUTH (default *MQADMIN block, no matching ADDRESSMAP). CONNAUTH failure (bad or missing password, CHKCLNT (REQUIRED)). OAM: the user/group has no CONNECT on the QM or no PUT/GET on the object.	Read the QM error log: <code>AMQ9777E/AMQ9776E = CHLAUTH</code> , <code>AMQ5534E = authentication</code> , <code>AMQ8077W = object authority</code> . Fix with a precise CHLAUTH rule, correct credentials, or SET AUTHREC + REFRESH SECURITY. Enable AUTHOREV for detail.
2009	MQRC_CONNECTION_BROKEN	A network drop or a firewall/load balancer idle timeout killed the TCP session. The QM ended or failed over. The channel instance was stopped.	Set HBINT (and TCP keepalive) lower than the firewall idle timeout. Turn on client auto-reconnect or retry logic. Check the QM log for the time of the failure.
2059	MQRC_Q_MGR_NOT_AVAILABLE	The QM is not running. Local bindings to a QM that does not exist on this host. Client: the listener is down, the CCDT or CONNAME is wrong, or the QM name does not match.	<code>dspmqr; DIS LSSTATUS(*)</code> ; check host(port), channel name and QM name. On clients, look further down the stack: the linked exception is often 2538 or a TLS error.
2538	MQRC_HOST_NOT_AVAILABLE	The client could not open a TCP connection to the host and port: no listener, firewall, DNS, wrong port.	<code>nc -vz host 1414</code> from the client; check listener status and port; check security groups and firewalls.
2085	MQRC_UNKNOWN_OBJECT_NAME	The queue does not exist on the QM the app is connected to. Wrong case (app.in vs APP.IN). The app is connected to the wrong QM. A cluster queue is not visible yet.	<code>DIS Q('NAME')</code> on the QM the app actually connects to; check case and quoting; for clusters, <code>DIS QCLUSTER('NAME')</code> .
2033	MQRC_NO_MSG_AVAILABLE	<i>Usually normal:</i> MQGET wait interval expired with no matching message. If unexpected: wrong queue, MsgId/CorrelId filter (often a stale MsgId reused from a previous get), messages not committed yet (UNCOM), messages expired.	Handle it as "no work" in poll loops. Reset MsgId/CorrelId to <code>MQMI_NONE/MQCI_NONE</code> between gets. Check <code>DIS QSTATUS . . . UNCOM</code> and the producer's commit logic.
2053	MQRC_Q_FULL	CURDEPTH reached MAXDEPTH: the consumer is down or slow, or a transmission queue is filling because its channel is not running.	Fix the consumer or channel first; raise MAXDEPTH only with disk headroom. Monitor with queue depth events (<code>QDEPTHHI/QDPHIEV</code>).
2537	MQRC_CHANNEL_NOT_AVAILABLE	The SVRCONN channel is STOPPED, or MAXINST/MAXINSTC/MAXCHL limit reached.	<code>DIS CHSTATUS(ch)</code> ; <code>START CHANNEL</code> ; check the instance limits against the connection count (connection leaks!).
2056	MQRC_Q_SPACE_NOT_AVAILABLE	The file system holding the queue file is full.	Free or extend <code>/var/mqm</code> . Then look for runaway queue depth.
2030 / 2031	MQRC_MSG_TOO_BIG_FOR_Q / _Q_MGR	The message is larger than MAXMSGL of the queue / the QM (and, over channels, the channel MAXMSGL).	Align MAXMSGL on the queue, QM, XMITQ and both channel ends, or segment / claim-check the payload.

2035 triage in 60 seconds: (1) `grep -E "AMQ9777|AMQ9776|AMQ5534|AMQ8077" /var/mqm/qmgrs/QM1/errors/AMQERR01.LOG` (2) `DIS CHLAUTH('CH') MATCH(RUNCHECK) ADDRESS('ip') CLNTUSER('user')` (3) `DIS CHSTATUS('CH') MCAUSER` to see the effective identity (4) `dspmqr -m QM1 -n QUEUE -t q -p user`.

The full pack covers 30 reason codes. The full 20-page *Kafka & IBM MQ Production Cheat Sheet Pack* (\$24) adds Kafka CLI, offset resets & reassignment, production configs, KRaft, MQSC, CHLAUTH/AUTHREC/CONNAUTH, channel troubleshooting, Kafka vs MQ vs RabbitMQ, MQ→Kafka migration notes and a pre-prod checklist.

kafka-mq-cheatsheet.pages.dev

Independent reference, not affiliated with or endorsed by the Apache Software Foundation, IBM or Broadcom. Test commands in a non-production environment first. Free to share this sample unchanged.

05 Kafka Consumer Lag & Rebalancing Checklist

Start with `kafka-consumer-groups.sh --describe --group G` and the `--members --verbose` view, then use the shape of the lag to find the cause.

Symptom	Likely causes	What to check / do
Lag rising evenly on all partitions	Consumers are under-provisioned. A slow downstream (DB, HTTP API). Records arrive faster than processing.	Compare the produce rate with the consume rate. Scale out instances (useful only up to the partition count). Batch downstream writes. Profile processing time per record. Add partitions only if you can tolerate the key remapping.
Lag on a few partitions only	Hot keys / skewed partitioner. One member is stuck or slow. One partition hits a slow or poison record.	Check per-partition produce rates (<code>kafka-get-offsets.sh</code> deltas). Use <code>--members --verbose</code> to see which member owns the lagging partitions. Take a thread dump of that instance.
Sawtooth lag + frequent rebalances	<code>max.poll.interval.ms</code> exceeded. Session timeouts from GC pauses or CPU throttling. Rolling deploys without static membership. Autoscaler flapping.	Search client logs for <i>"consumer poll timeout has expired"</i> and <code>CommitFailedException</code> . Lower <code>max.poll.records</code> . Set <code>group.instance.id</code> . Use <code>cooperative-sticky</code> or <code>group.protocol=consumer</code> .
Lag frozen, consumers "healthy"	A poison record retried forever. A <code>read_committed</code> consumer blocked by an open or hanging transaction (LSO stuck). Commits failing.	Read the record at the stuck offset with the console consumer. Look for hanging transactions: <code>kafka-transactions.sh --bootstrap-server \$B find-hanging --broker-id 1</code> . Check commit errors and ACLs (Read on the group).
Data skipped / offsets jumped	Committed offsets expired while the group was empty (<code>offsets.retention.minutes</code> , default 7 days) and <code>auto.offset.reset=latest</code> . The consumer fell behind retention (<code>OffsetOutOfRange</code>).	Alert on lag measured in <i>time</i> as well as in records. Make sure retention $\geq$ the maximum expected outage. Set <code>auto.offset.reset</code> explicitly.
Group stuck in PreparingRebalance	A member is not rejoining (blocked in processing or a long <code>onPartitionsRevoked</code> callback).	Keep rebalance listeners fast. Make sure processing threads stop promptly when partitions are revoked. In classic groups, the rebalance waits up to <code>max.poll.interval.ms</code> for members to rejoin.

Common rebalance triggers

- Member join/leave: deploys, scaling, crashes, OOM kills
- No heartbeat within the session timeout (network, GC, CPU starvation)
- `poll()` not called within `max.poll.interval.ms`
- Subscription change (regex picks up a new topic) or partition count change
- Group coordinator moving (the broker hosting that `__consumer_offsets` partition restarts)

Client metrics worth graphing

Metric (consumer)	Signal
<code>records-lag-max</code>	Worst partition lag for this client
<code>time-between-poll-max</code>	Getting close to <code>max.poll.interval.ms</code> ?
<code>last-poll-seconds-ago</code>	Poll loop stalled
<code>rebalance-rate-per-hour</code>	Rebalance storms
<code>commit-latency-avg</code>	Slow coordinator / broker

Fix patterns

- Bounded poll work:** `max.poll.records` $\times$ p99 time per record should stay well below `max.poll.interval.ms`.
- Static membership** (`group.instance.id`): restarts shorter than the session timeout cause no rebalance at all.
- Incremental rebalancing:** `CooperativeStickyAssignor` (classic) or the KIP-848 `consumer` protocol, so unaffected partitions keep processing.
- Dead-letter topic:** after N retries, send the poison record (with error headers) to `<topic>.DLT` and commit past it.
- Parallelism beyond partitions:** key-ordered worker pools inside one consumer, or share groups (KIP-932, production-ready in Kafka 4.2) when per-partition ordering is not required.
- Commit discipline:** commit only after side effects finish, and commit synchronously in `onPartitionsRevoked`.

```
# Abort a hanging transaction blocking read_committed consumers
kafka-transactions.sh --bootstrap-server $B find-hanging -
-broker-id 1
kafka-transactions.sh --bootstrap-server $B abort \
--topic orders --partition 3 --start-offset 991204
```

Want the rest? The full 20-page *Kafka & IBM MQ Production Cheat Sheet Pack* (\$24) adds Kafka CLI, offset resets & reassignment, production configs, KRaft, MQSC, CHLAUTH/AUTHREC/CONNAUTH, channel troubleshooting, Kafka vs MQ vs RabbitMQ, MQ→Kafka migration notes and a pre-prod checklist.

kafka-mq-cheatsheet.pages.dev