Apache Kafka Interview Guide (100+ Questions)

Below is a structured guide with 4 sets, each containing 25 Kafka interview questions. Each question includes an explanation or answer.

Set 1: Core Kafka Concepts (25 Questions)

1. **What is Apache Kafka?** Kafka is a distributed event-streaming platform used for building real-time data pipelines and streaming applications.

2. **What is a Topic?** A category or stream name to which records are stored.

3. **What is a Partition?** A topic is split into partitions for parallelism and scalability.

4. **What is an Offset?** A unique ID assigned to each message within a partition.

5. **What is a Consumer Group?** A group of consumers working together to consume messages from a topic.

6. **What is a Producer?** An application that publishes messages to Kafka topics.

7. **What is a Broker?** A Kafka server storing data and serving client requests.

8. **What is Zookeeper?** Used by Kafka (pre-Kraft) for broker coordination.

9. **What is the use of Kafka Controller?** Manages partition leadership and replication.

10. **What is Message Retention?** How long Kafka stores messages (time-based or size-based).

11. **What is Log Compaction?** Kafka keeps only the latest value for each key.

12. **What is a Replica?** A copy of partition data stored across brokers.

13. **What is ISR (In-Sync Replica)?** A set of replicas that are fully caught up with the leader.

14. **What is ACK in Kafka?** Defines how many replicas must acknowledge a write.

15. **Difference between `acks=0`, `acks=1`, `acks=all`**

16. 0 → worst durability, fastest
17. 1 → leader-only acknowledgment

18. all → safest, slowest

19. **What is Kafka Streams?** A library for building streaming applications.
20. **What is Exactly-Once Semantics?** Kafka guarantees a message is processed exactly once.
21. **What is Consumer Lag?** Difference between last produced and last consumed message.
22. **What are the main components of Kafka?** Producers, Consumers, Brokers, Topics, Partitions.
23. **What is a Dead Letter Queue?** A topic for failed or unprocessable messages.
24. **What is Rebalancing?** Redistribution of partitions across consumers.
25. **What is Sticky Partitioning?** Producer sends messages to the same partition until batch full.
26. **What are Records?** The actual key-value messages stored in Kafka.
27. **What is Kafka Connect?** A tool to transfer data between Kafka and external systems.
28. **Use of Schema Registry** Manages schema versions for Kafka messages.

Set 2: Message Consumption & Consumer Errors (25 Questions)

1. **How does a consumer read messages?** It polls data from Kafka using `poll()` API.
2. **What happens when two consumers are in the same consumer group?** Kafka divides partitions between them.
3. **What if partitions < consumers?** Extra consumers remain idle.
4. **What if partitions > consumers?** Some consumers handle multiple partitions.
5. **Why messages remain unconsumed?** Consumer down, partition mismatch, lag, wrong group ID.
6. **What causes consumer lag?** Slow processing, network issues, insufficient consumers.
7. **Fix consumer lag** Increase partitions, scale consumers, optimize processing.
8. **What is commit offset?** Marks messages as processed.
9. **What is auto commit?** Kafka automatically commits offsets.
10. **When does auto commit fail?** If consumer crashes before commit.
11. **Manual commit advantages** Control over message acknowledgement.

12. **What happens if consumer fails after processing but before commit?** Message will be reprocessed → at-least-once.

13. **What is at-most-once processing?** Messages may be lost.

14. **What is at-least-once?** Messages may be duplicated.

15. **Errors:** `OffsetOutOfRangeException` Offset deleted due to retention. Fix → reset offset = earliest/latest.

16. `RebalanceInProgressException` Occurs during consumer group rebalance. Fix → handle commit in try/catch.

17. `CommitFailedException` Commit attempted after rebalance. Fix → retry commit.

18. `SerializationException` Invalid message format. Fix → correct serializer/deserializer.

19. `TimeoutException` **while consuming** Slow broker or network. Fix → increase `poll.timeout`.

20. **What is max.poll.interval.ms?** Max time between polls.

21. **What happens when max.poll.interval exceeded?** Kafka removes consumer from group.

22. **What is max.poll.records?** Max messages returned per poll.

23. **What is heartbeat interval?** Prevents consumer removal from group.

24. **Why consumer stuck in rebalancing?** Slow heartbeat, overloaded consumer.

25. **Fix rebalancing loop** Tune: heartbeat, max.poll.interval, session.timeout.

Set 3: Partitions, Scaling & Producer Errors (25 Questions)

1. **How are partitions assigned?** Round-robin, sticky, or custom partitioner.

2. **What happens when you increase partitions?** Parallelism increases but message order breaks.

3. **Can you decrease partitions?** No → irreversible.

4. **What is message ordering guarantee?** Kafka guarantees ordering only within a partition.

5. **How to ensure ordering?** Use key-based partitioning.

6. **Producer** `BufferExhaustedException` Insufficient buffer size.
7. **Producer** `TimeoutException` Brokers overloaded.
8. **What is idempotent producer?** Prevents duplicates.
9. **Use of** `enable.idempotence=true` Guarantees exactly-once for producers.
10. **What is transactional producer?** Used for atomic multi-partition writes.
11. **What is linger.ms?** Delay before sending batch to accumulate messages.
12. **What is batch.size?** Max message batch size.
13. **What is compression.type?** Snappy, gzip, zstd reduce payload size.
14. **What is replication.factor?** Number of replicas for durability.
15. **What if replication factor > brokers?** Topic creation fails.
16. **Leader election in Kafka** Controller assigns partition leaders.
17. **Under-replicated partition?** Some replicas not in sync.
18. **Fix under-replicated partitions** Check broker down, network issues.
19. **Unclean leader election** Allows out-of-sync replica to become leader → data loss.
20. **min.insync.replicas** Minimum replicas required for write.
21. **Producer** `RecordTooLargeException` Message size exceeds limit.
22. **Fix message too large** Increase `max.request.size`, `message.max.bytes`.
23. **What is retention.ms?** Time-based message retention.
24. **What is retention.bytes?** Size-based log retention.
25. **What is segment.ms?** Time to roll log segment.

Set 4: Real-World Scenarios & Tricky Questions (25 Questions)

1. **Two consumers reading same topic but receiving same messages?** They have different consumer groups.

2. **Two consumers, same group, same partition?** Impossible; one partition assigned to only one consumer.

3. **Consumer not receiving messages after restart** Committed offset points to latest.

4. **How to reprocess all messages?** Reset offset → earliest.

5. **How to handle poison messages?** Send to Dead Letter Queue.

6. **Kafka losing messages?** Possible if:

7. acks=0 or acks=1
8. unclean leader election
9. low replication

10. **How to guarantee no message loss?** acks=all, replication>=3, idempotent producer.

11. **Why unconsumed messages increase?** Consumer lag.

12. **Consumer processed message but crashed before commit** Reprocessed.

13. **Producer retry logic** Uses exponential backoff.

14. **What if producer retries cause duplicates?** Enable idempotence.

15. **Partition leader down** ISR replica becomes new leader.

16. **What if no ISR available?** Cluster unavailable.

17. **How Kafka achieves scalability?** Partitioning + replication.

18. **Kafka as queue vs. pub/sub** Queue → one consumer per partition Pub/Sub → multiple consumer groups

19. **Kafka vs RabbitMQ** Kafka → streaming RabbitMQ → messaging

20. **Kafka vs ActiveMQ** Kafka for high throughput.

21. **What is backpressure?** Consumer slower than producer.

22. **Fix backpressure** Scale consumers, increase partitions.

23. **What is watermarking in Kafka Streams?** Marks event time progress.

24. **What is windowing?** Aggregations over time intervals.

25. **What is retention vs compaction?** Retention deletes old data; compaction keeps latest.

26. **Can Kafka lose data after commit?** Rare, unless disk corruption.

27. **What is replication throttle?** Limits replication bandwidth.

28. **Kafka exactly-once in distributed system** Uses transactions + idempotent writes.

Additional Sections Added (Cheat-sheets, Component Guides, Advanced Topics, Tricky Scenarios)

Cheat Sheet — Commands, Useful Scripts & Quick Ops

Common CLI commands (Kafka binary distribution)

- List topics: `kafka-topics.sh --bootstrap-server <broker:9092> --list`
- Describe topic: `kafka-topics.sh --bootstrap-server <broker:9092> --describe --topic <topic-name>`
- Create topic: `kafka-topics.sh --bootstrap-server <broker:9092> --create --topic <topic-name> --partitions <N> --replication-factor <R>`
- Delete topic: `kafka-topics.sh --bootstrap-server <broker:9092> --delete --topic <topic-name>`
- Produce to console: `kafka-console-producer.sh --broker-list <broker:9092> --topic <topic-name>`
- Consume from console: `kafka-console-consumer.sh --bootstrap-server <broker:9092> --topic <topic-name> --from-beginning`
- Consumer group list: `kafka-consumer-groups.sh --bootstrap-server <broker:9092> --list`
- Consumer group describe: `kafka-consumer-groups.sh --bootstrap-server <broker:9092> --describe --group <group-id>`
- Reset offsets: `kafka-consumer-groups.sh --bootstrap-server <broker:9092> --group <group-id> --reset-offsets --to-earliest --topic <topic> --execute`
- Reassign partitions (create JSON then run): `kafka-reassign-partitions.sh --bootstrap-server <broker:9092> --reassignment-json-file reassignment.json --execute`
- Preferred leader election: `kafka-preferred-replica-election.sh --bootstrap-server <broker:9092>`
- Tool to delete records (log-retention workaround): `kafka-delete-records.sh --bootstrap-server <broker:9092> --offset-json-file offsets.json --execute`

Quick config lookups

- Broker config: `kafka-configs.sh --bootstrap-server <broker:9092> --entity-type brokers --entity-name <broker-id> --describe`
- Topic-level config: `kafka-configs.sh --bootstrap-server <broker:9092> --entity-type topics --entity-name <topic-name> --describe`

Scripts you should keep handy

- Health-check script: checks broker port 9092, ZK/KRaft status, under-replicated partitions, and disk usage.
- Rebalance-safe-deploy script: pause consumers, increase partitions (if needed), run reassignments, then resume consumers.

Cheat Sheet — Important Configs (and recommended starting values)

Broker (server.properties)

- `num.network.threads` — default 3. Increase with high throughput.
- `num.io.threads` — default 8. Increase if high disk I/O.
- `log.dirs` — multiple mount points recommended.
- `num.partitions` — default partition count for new topics (e.g., 3–12 depending on use-case).
- `default.replication.factor` — recommended 3 for production.
- `log.segment.bytes` — default 1GB; decrease for faster recovery if required.
- `log.retention.hours` / `log.retention.bytes` — retention policy.
- `replica.fetch.max.bytes` & `message.max.bytes` — ensure producer max < broker limits.
- `unclean.leader.election.enable` — set `false` for production durability.
- `min.insync.replicas` — set to 2 for replication factor 3.

Producer

- `acks=all` — recommended for durability.
- `retries` — set >0 and combine with `enable.idempotence=true`.
- `enable.idempotence=true` — avoid duplicates.
- `max.in.flight.requests.per.connection` — set 1 or leave default 5 (with idempotence, safe to keep >1 in modern Kafka).
- `linger.ms` — 0–100ms (higher batches throughput but adds latency).
- `batch.size` — tune for throughput; e.g., 32KB–512KB.
- `compression.type` — `snappy` or `lz4` for low CPU + good compression.
- `max.request.size` — ensure larger than biggest message.

Consumer

- `group.id` — unique consumer group id.
- `enable.auto.commit=false` — manual commit recommended in most use-cases.

- `auto.offset.reset` — `latest` or `earliest` depending on requirement.
- `max.poll.records` — control per-poll load (e.g., 500–2000).
- `max.poll.interval.ms` — time allowed for processing (increase if processing is slow).
- `session.timeout.ms` & `heartbeat.interval.ms` — tune for network/GC pauses. `heartbeat.interval.ms` should be < `session.timeout.ms/3`.

Connect / Streams

- `offset.storage.topic` / `config.storage.topic` / `status.storage.topic` — monitor their replica counts and partitions.
- Streams: `processing.guarantee=exactly_once_v2` (or `exactly_once`) for exactly-once.

Tuning parameters — quick heuristics

- Start with `replication.factor=3`, `min.insync.replicas=2`, `acks=all`.
- For high throughput increase partitions per topic; measure consumer CPU/memory.
- Balance `linger.ms` and `batch.size` for producer latency vs throughput.
- Tune JVM GC and avoid long GC pauses — these cause long consumer heartbeats and rebalances.

Separate Guides: Producer, Consumer, Broker, Topic, Partition

Producer — Quick Guide

Core responsibilities: serialize and send messages to topics.

Essential configs & why: - `bootstrap.servers` — initial broker list to connect. - `key.serializer` / `value.serializer` — must match deserializers at consumer. - `acks` — durability tradeoff. - `retries` + `enable.idempotence` — avoid duplicates when retrying. - `transactional.id` + `initTransactions()` — for transactions across partitions/topics.

Patterns & best practices - Use keys for partitioning when ordering matters. - Keep messages small; larger messages affect broker throughput. - Use compression to reduce bandwidth. - Catch and handle `SerializationException`, `TimeoutException`, `NetworkException`.

Common interview traps - Misunderstanding idempotence vs transactions: idempotence prevents duplicates from retries for single producer instance; transactions provide atomic writes across partitions/topics and consumer offsets. - `max.in.flight.requests.per.connection` with retries — historically could cause out-of-order writes; idempotence now helps.

Troubleshooting - `RecordTooLargeException` → increase `max.request.size` and `message.max.bytes`. - High latency → increase `batch.size` and `linger.ms`, check broker CPU/disk.

Consumer — Quick Guide

Core responsibilities: poll, process, and commit offsets.

Key concepts to master: - Poll loop design: poll, process batch, commit offsets. - `enable.auto.commit=false` is safer; commit after successful processing. - At-least-once vs at-most-once vs exactly-once (with Kafka transactions and Streams). - Consumer group rebalancing and its impact.

Important settings: - `max.poll.records` and `max.poll.interval.ms` — affects rebalances. - `heartbeat.interval.ms` and `session.timeout.ms` — tradeoff between detection speed and robustness to pauses.

Common pitfalls & fixes: - Long processing inside poll loop causing `CommitFailedException` or `RebalanceInProgressException` → process outside critical section, reduce `max.poll.records`, increase `max.poll.interval.ms`. - Offset commit after processing but before external side-effect leads to duplication if crash occurs — use transactions or idempotent sinks.

Diagnostics: - `kafka-consumer-groups.sh --describe` to check lag. - Broker logs and controller logs for rebalances and group coordinator messages.

Broker — Quick Guide

Core responsibilities: storage, replication, and serving client requests.

Operational concerns: - Disk usage and `log.dirs` — keep logs on fast disks and monitor retention. - JVM tuning: GC is critical — avoid long GC pauses as they cause disconnects. - Network: ensure sufficient NIC bandwidth and thread pools. - Monitoring: under-replicated partitions, offline partitions, broker leader counts.

Key configs: `num.network.threads`, `num.io.threads`, `socket.receive.buffer.bytes`, `socket.send.buffer.bytes`, `replica.fetch.max.bytes`, `replica.fetch.wait.max.ms`.

High-availability best practices: - Spread replicas across availability zones. - Run odd number of brokers to avoid quorum issues. - Set `unclean.leader.election.enable=false` in production.

Backup & Recovery: - Snapshotting isn't built-in; rely on replication and proper retention policies. Use MirrorMaker or cluster replication for DR.

Topic — Quick Guide

Core responsibilities: logical namespace for messages.

Topic lifecycle commands: create, alter configs, delete, describe.

Important configs per-topic: `retention.ms`, `retention.bytes`, `cleanup.policy` (`delete` or `compact`), `segment.bytes`, `min.insync.replicas`.

Design considerations: - Choose partitions based on consumer throughput needs and parallelism. - Use compaction for state-change logs (e.g., changelog topics for KTable). - Be careful increasing partitions after data already in place — ordering guarantees are per-partition and will be affected.

Partition — Quick Guide

Core responsibilities: unit of parallelism and ordering.

Key concepts: leader, followers, ISR, replica, partition assignment to brokers.

Operational notes: - Monitor leader distribution — avoid broker hotspots. - Reassignment can be online but will re-balance data across disks and network. - Under-replicated partitions indicate problems — check broker health and replication settings.

Interview traps: - "Can we reduce partition count?" — No, you can increase but not decrease safely. - "Does partitioning ensure global ordering?" — No, only per-partition.

Advanced Kafka Streams & Connect Interview Section

Kafka Streams — Core topics to know

- Topology: `KStream`, `KTable`, `GlobalKTable`.
- Stateless vs stateful operations (filter, map vs aggregations, joins, windows).
- State stores: RocksDB-backed local state stores and changelog topics.
- Repartitioning topics: when you change key you may need a repartition step and topic.
- Time semantics: event-time vs processing-time vs ingestion-time; watermarks for late-arriving events.
- Exactly-once semantics: `processing.guarantee=exactly_once_v2` and how it works with EOS v2 (idempotent writes + transactional offset commits in Streams)
- Interactive queries: exposing local state stores to serve reads.
- Scaling & fault tolerance: partition-to-task assignment; tasks migrate on rebalancing.

Common tricky questions - Difference between `KTable` and `GlobalKTable` and when to use each. - How state stores get restored after failure (changelog topic consumption during task startup). - How to do joins across topics with different partition counts.

Kafka Connect — Core topics to know

- Source vs Sink connectors.
- Connect workers: standalone vs distributed mode.

- Connector configs: `tasks.max`, `topic.creation.configs`, converters (Avro/JSON/String), transforms (SMTs).
- Schema handling: Avro + Schema Registry, or JSON Schema.
- Exactly-once sink semantics: dependent on connector capabilities and sink idempotency.

SMTs (Single Message Transforms) - Simple per-record transforms: `InsertField`, `MaskField`, `SetSchemaMetadata`, `ReplaceField`. - Use-case: drop fields, rename fields, or route messages.

Troubleshooting Connect - Connector tasks failing repeatedly → check converter errors, schema mismatch. - Offsets topic full or misconfigured → ensure correct partitions/replication for internal topics.

More Tricky Scenario-Based Questions (20 scenarios + answers)

1. **Scenario: Two consumers in different groups receive the same messages. Why?**
2. Explanation: Consumers in different groups each receive full topic streams — this is expected in pub/sub semantics.
3. **Scenario: Two consumers in the same group receiving duplicate messages.**
4. Possible causes: Offsets not committed correctly, manual commit bugs, or processing after commit causing replays. Fix by ensuring commit occurs after processing and/or use transactions.
5. **Scenario: Consumer keeps rebalancing in a loop.**
6. Causes: long GC pauses, `max.poll.interval.ms` too low, heartbeat issues, or frequent group coordinator restarts. Fix: raise `max.poll.interval.ms`, tune GC, increase heartbeat frequency.
7. **Scenario: Producer gets `RecordTooLargeException` for a message that fits in broker `message.max.bytes`.**
8. Likely broker and topic config mismatch (`message.max.bytes` vs `replica.fetch.max.bytes` vs `max.request.size`). Fix by aligning producer and broker configs.
9. **Scenario: Messages are produced but consumers see no new data (lag increasing on producer side but consumers show no increase).**
10. Could be producers writing to different cluster/topic name, ACLs preventing consumers, or consumer group misconfiguration (different group id). Use `kafka-topics.sh --describe` to verify partitions and offsets.
11. **Scenario: OffsetOutOfRangeException on consumer startup.**
12. Caused when committed offset is older than earliest retained offset. Fix by resetting offsets to earliest/latest using `kafka-consumer-groups.sh` or set `auto.offset.reset=earliest`.

13. **Scenario: Under-replicated partitions after a broker restart.**

14. Possible slow follower replication or broker down. Fix broker start, check `replica.fetch.*` limits, network.

15. **Scenario: High end-to-end latency.**

16. Causes: large `linger.ms`, overloaded brokers, GC pauses, network saturation. Fix: tune batching, resize cluster, improve JVM/GC.

17. **Scenario: Messages disappear after a broker leader failover.**

18. Caused by `unclean.leader.election.enable=true` — an out-of-sync replica became leader. Disable unclean election.

19. **Scenario: Consumer sees messages but processing fails sometimes (poison pill).**

20. Detect and route failing records to DLQ (Dead Letter Queue) after N retries using retry topics.

21. **Scenario: Transactional producer writes but consumer doesn’t see messages.**

22. When consumers use `isolation.level=read_committed` they won't see uncommitted transactional messages — ensure transactions are committed.

23. **Scenario: Kafka Connect connector consumes but sink doesn't receive data.**

24. Check connector task logs, converter errors (schema mismatch), and sink connector configs (ACLs, connectivity).

25. **Scenario: Slow replica sync causing ISR shrink.**

26. Network/disk I/O problems on follower broker. Investigate disk throughput and network.

27. **Scenario: Consumer lag grows in bursts.**

28. Likely GC pauses on consumer, or periodic heavy downstream processing. Profile consumer processing.

29. **Scenario: Topic with compact policy grows large unexpectedly.**

30. Compaction keeps latest key versions; if unique keys keep increasing compaction won't reduce size — check key cardinality.

31. **Scenario: Reassign partitions causes performance hit.**

32. Migration moves data across brokers; throttle replication using `replication.quota.window.num` and `quota` settings.

33. **Scenario: Streams app state store not restored after rebalance.**

34. Ensure changelog topics exist, have sufficient replication, and correct partitions; inspect logs for state restore errors.

35. **Scenario: Connect offsets or configs topic corrupted.**

36. Restore from backups or recreate internal topics and restart workers; ensure `config.storage.replication.factor` >= 3.

37. **Scenario: You need to reprocess historical data but also keep current consumers running.**

38. Create a new consumer group id to reconsume from `earliest`, or mirror topic to a new topic and process there.

39. **Scenario: Exactly-once guarantees failing in Streams app after upgrade.**

40. Check `processing.guarantee` settings, transactional IDs compatibility, and if EOS v2 semantics configuration is correct.

I've added all new sections above to this document: cheat sheets, separate component guides, advanced Kafka Streams & Connect topics, and 20 new tricky scenarios with fixes. You can ask me to:

- Expand any single section into deeper sub-sections (example: full Streams failure-recovery playbook).
- Convert the doc into PDF or downloadable file.
- Add diagrams or example `server.properties`, `producer.properties`, and sample code snippets.

End of document.