READ MODE · ANSWER FIRST

How Kafka Consumer Groups Actually Rebalance

An interactive, visual explanation of How Kafka Consumer Groups Actually Rebalance, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

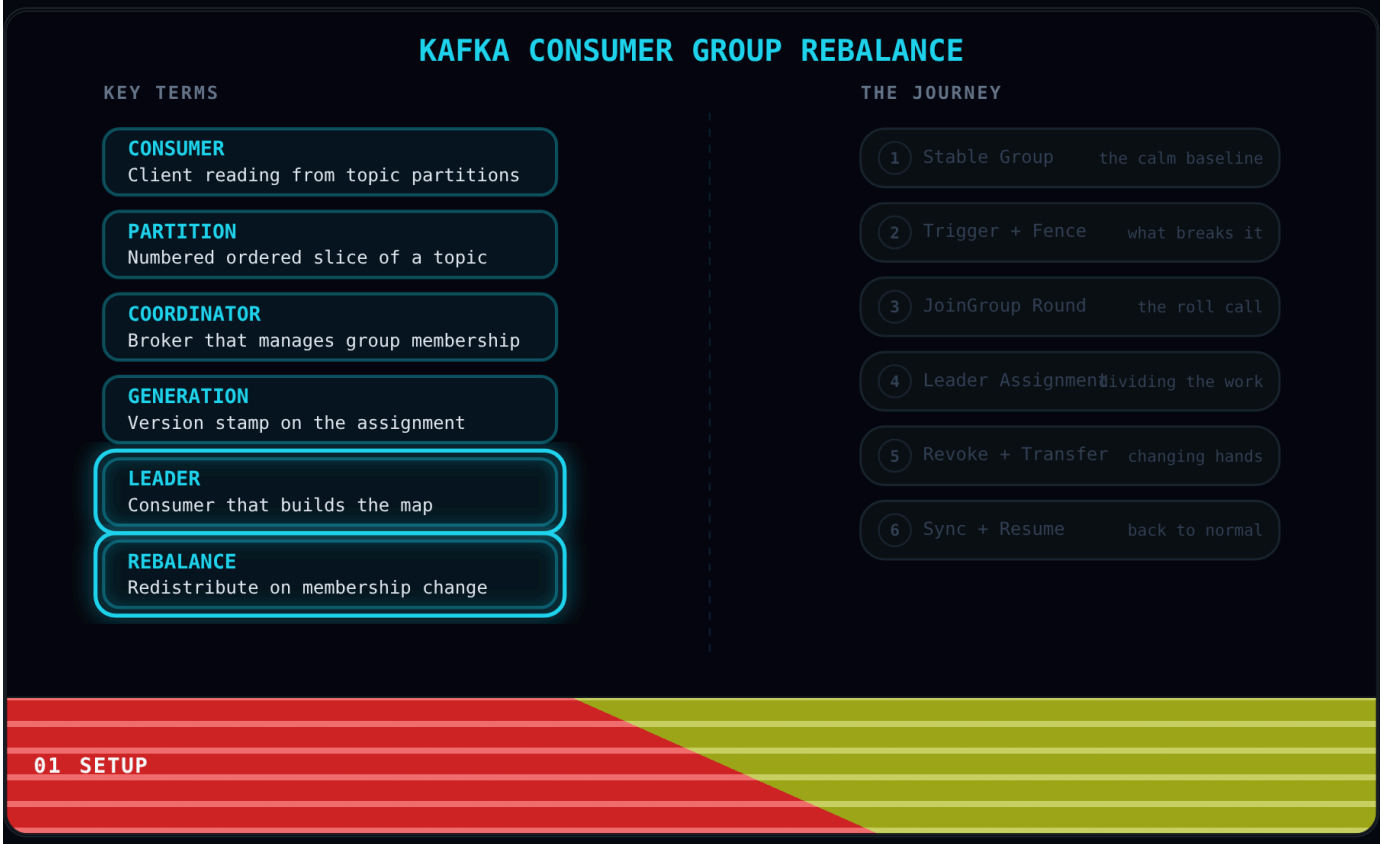
The whole story in 6 lines

How Kafka Consumer Groups Actually Rebalance becomes easier to reason about when every stage is connected as one system.

1. Teach the baseline steady-state before anything breaks: coordinator heartbeats, one elected leader, and partition ownership that looks...
2. Make the trigger visible as a generation fence, not a mysterious black box.
3. Show JoinGroup as a rendezvous: the coordinator waits for member identities, subscriptions, and protocols, then chooses one leader for...
4. This stage should make the assignor feel concrete: range groups contiguous partitions, round robin stripes them, sticky minimizes movement.
5. Teach the operational difference: eager drops everything at once, cooperative only revokes partitions that truly need to move.
6. Close the loop: rebalance finishes only after SyncGroup, local install, and resumed fetches on the new generation.

STAGE 1 OF 8

Setup



Welcome. Kafka consumer groups let multiple instances of your application share the work of reading from a topic. When membership changes, partitions must be redistributed. That redistribution is called a rebalance, and understanding how it works is the key to avoiding surprises in production.

A consumer is a client that reads messages from a Kafka topic. A partition is a numbered slice of that topic that stores messages in strict order. Think of partitions as lanes in a highway and consumers as the cars driving in those lanes.

A coordinator is the broker responsible for managing who belongs to the group. A generation is a version number stamped on the current assignment. Every time ownership changes, the generation ticks up and invalidates the old map.

A leader is the one consumer chosen by the coordinator to compute the partition assignment. Rebalance is the full stop and redistribute process that fires whenever membership changes.

These six terms will appear throughout every stage. Notice how they connect: the coordinator tracks the generation, the leader builds the assignment, and the consumers own the partitions. Together they form the machinery of a rebalance.

Over the next six stages we will walk through the full lifecycle: what a healthy group looks like, what triggers a rebalance, how members gather, how work is divided, how partitions change hands, and how everything stabilizes again. Let us start with the calm before the storm.

STAGE 2 OF 8

Stable Group

02 STABLE GROUP

This is what a healthy consumer group looks like. Every partition already has an owner and every consumer is heartbeating under the current generation. Nothing dramatic is happening yet.

Consumers spend most of their time doing ordinary fetches from their assigned partitions. Data flows steadily from partitions to consumers. Try changing the Members control in the sidebar to see how partition ownership distributes across different group sizes.

Heartbeats flow in the opposite direction. Each consumer sends periodic heartbeats to the coordinator to prove it is still alive and valid for this generation.

One consumer is marked as leader, but while the group is stable the leader is not actively computing anything. The leader role only becomes important when a rebalance triggers.

The generation number is the coordinator's stamp of authority. It says: this specific assignment of partitions to consumers is the current truth. Any change to membership will fence this generation.

This calm mapping is what a rebalance has to tear down and rebuild correctly. Next, we will see what kinds of events break this stability and force the group to reorganize.

STAGE 3 OF 8

Trigger + Fence

03 TRIGGER + FENCE

We just saw what a stable group looks like. Now something changes. A new consumer asks to join, an existing one goes silent, or the topic gains new partitions. Any of these events forces the coordinator to stop trusting the current generation.

Switch the Cause control in the sidebar between Join, Leave, and Expand. Notice how each trigger sends a different signal to the coordinator. A join means a new member wants in. A leave means a heartbeat went missing. An expand means the topic itself changed shape.

The coordinator treats that event as a fence on the old generation. Fencing is the real start of a rebalance. It means the prior ownership map can no longer be trusted, even though data might still be flowing.

Once the generation is fenced, consumers cannot keep polling under the stale assignment. The coordinator will reject heartbeats that reference the old generation.

Polling pauses and the entire group shifts from steady fetches into rebalance mode. Every consumer must rejoin, not just the one that triggered the change.

At this point no one owns any partitions for certain. The system is forcing everyone to gather and renegotiate. Next, we will see how that gathering happens through the JoinGroup protocol.

STAGE 4 OF 8

JoinGroup Round

04 JOINGROUP ROUND

Now that the old generation is fenced, Kafka needs to gather every valid member into one specific rebalance attempt. This is the JoinGroup round. Think of it like a roll call before a team huddle.

Each member sends a JoinGroup request to the coordinator with its identity, topic subscriptions, and supported rebalance protocol. Watch the messages flow from consumers to the coordinator.

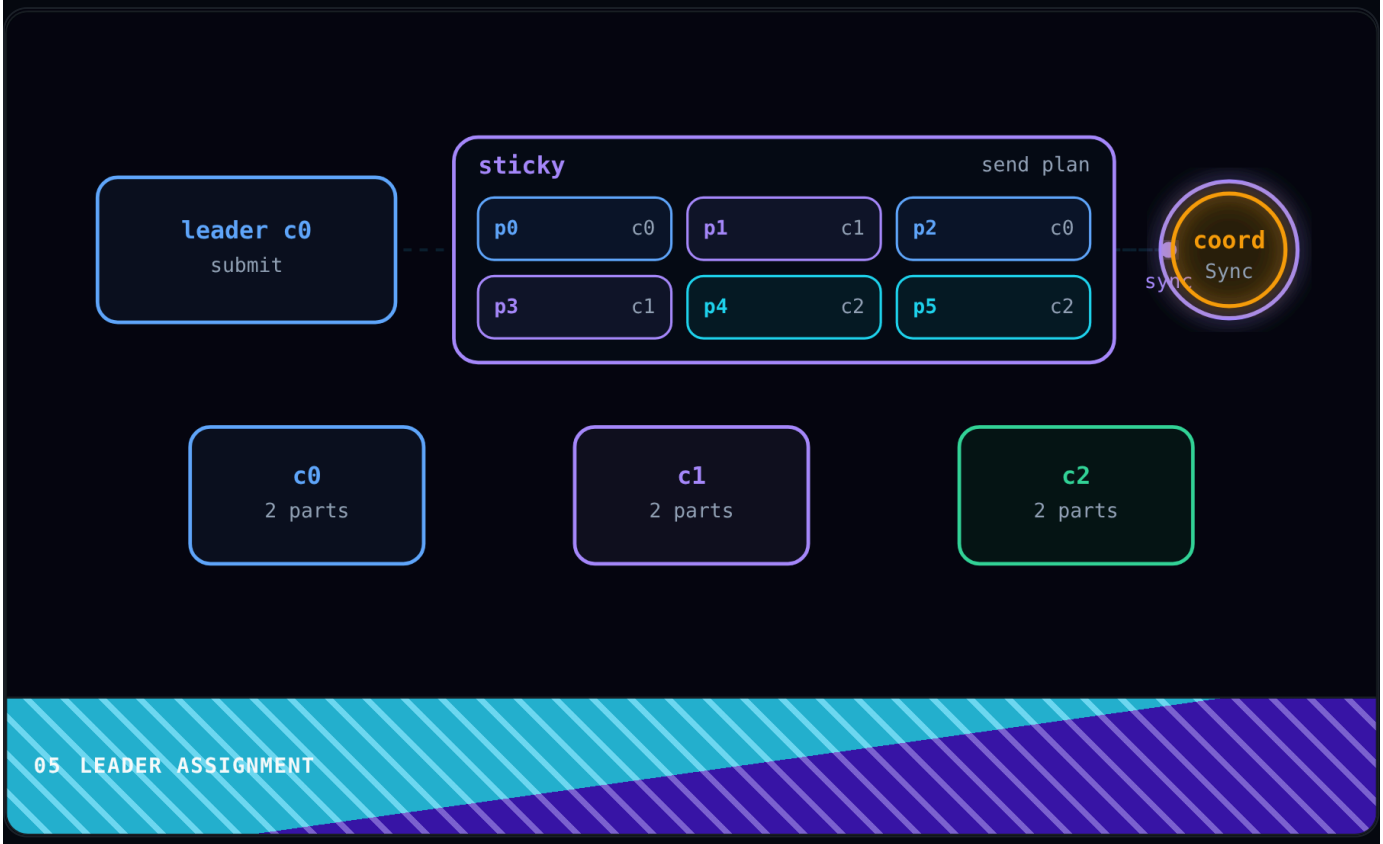
The coordinator waits until the group is complete enough to proceed. It collects all join requests and builds one pending generation from the full roster. Try changing the Members control to see how more participants look in the window.

Kafka elects one member as leader for this specific round. The leader is not a permanent role. It is just the member that Kafka picks to compute the assignment for this particular rebalance.

JoinGroup responses fan back to every member so they all learn who the leader is and which rebalance protocol was selected. The Protocol control lets you see the difference between eager and cooperative labels.

The group now has a coherent set of participants and a designated leader, but still no partition map. That computation happens next when the leader turns the member list into an actual assignment.

Leader Assignment



05 LEADER ASSIGNMENT

We just learned who the leader is. Now the leader has to do its job: take the member list plus topic metadata and compute a partition map. This is where the assignor strategy becomes visible.

The leader starts with the full set of joined members and the current partition metadata for the subscribed topics. These are the inputs to the assignment algorithm.

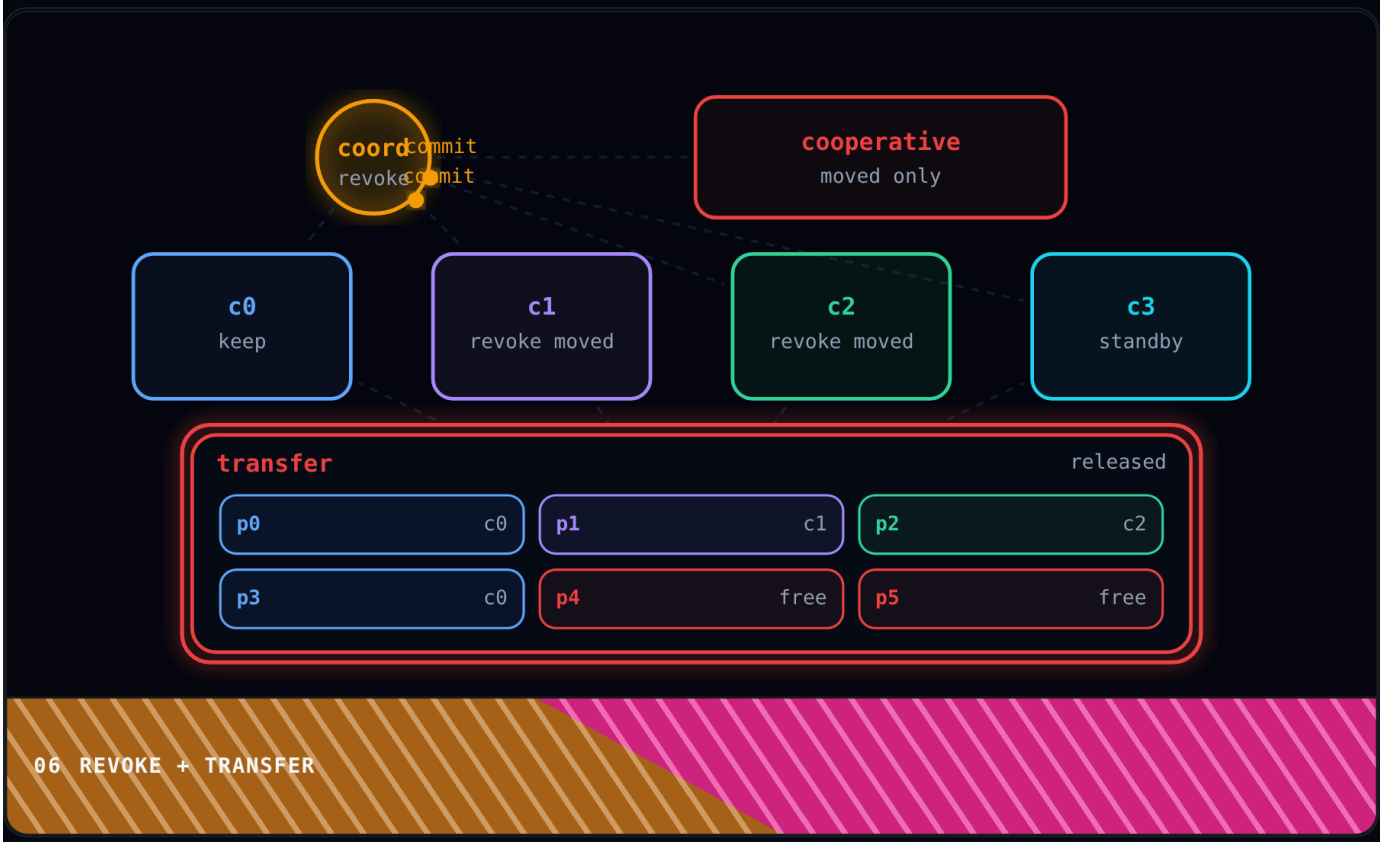
Switch the Assignor control between Range, RoundRobin, and Sticky. Range gives contiguous blocks. RoundRobin stripes partitions across members one by one. Sticky tries to keep prior owners whenever possible to minimize movement.

The leader packages the computed map into a SyncGroup request and sends it to the coordinator. This is the only moment where the assignment exists outside the leader's memory.

The coordinator validates the proposal and associates it with the pending generation. The plan now has a home on the broker side.

The rebalance now has a new owner map on paper. But a plan is not the same as reality. The remaining question is how partitions actually change hands. That is the hardest part, and it comes next.

Revoke + Transfer



06 REVOKE + TRANSFER

Remember the assignment we just computed? Now it has to take effect. That means some partitions need to move from old owners to new ones. This is the stage where the real operational cost of a rebalance becomes visible.

The coordinator identifies which partitions must move to satisfy the new assignment. The Moving control in the sidebar lets you adjust how many partitions are changing hands.

Switch the Protocol control between Eager and Cooperative. With eager rebalance, every partition is revoked from every consumer, even the ones not moving. With cooperative, only the partitions that actually need to move are revoked.

Source owners commit their current offsets, drain any in-flight messages, and release the partitions that are changing hands. This is the drain phase.

Only after the old owner has fully released a partition can the target owner claim it. The handoff must be clean. No partition can have two active owners at the same time.

This transfer phase explains why cooperative rebalance matters in production. Eager mode stops everything. Cooperative mode stops only what has to move. Now let us see how the group finishes the job and gets back to normal.

Sync + Resume

07 SYNC + RESUME

We just saw how partitions change hands. Now the coordinator needs to confirm the new assignment with everyone, heartbeats need to switch to the new generation, and consumers need to resume polling. This is the finish line.

SyncGroup responses fan out from the coordinator to every member. Each consumer receives its personal slice of the new partition map.

Members install the assignment locally, seek to the correct offsets in their new partitions, and prepare to resume. This local installation is what makes the plan real on each consumer.

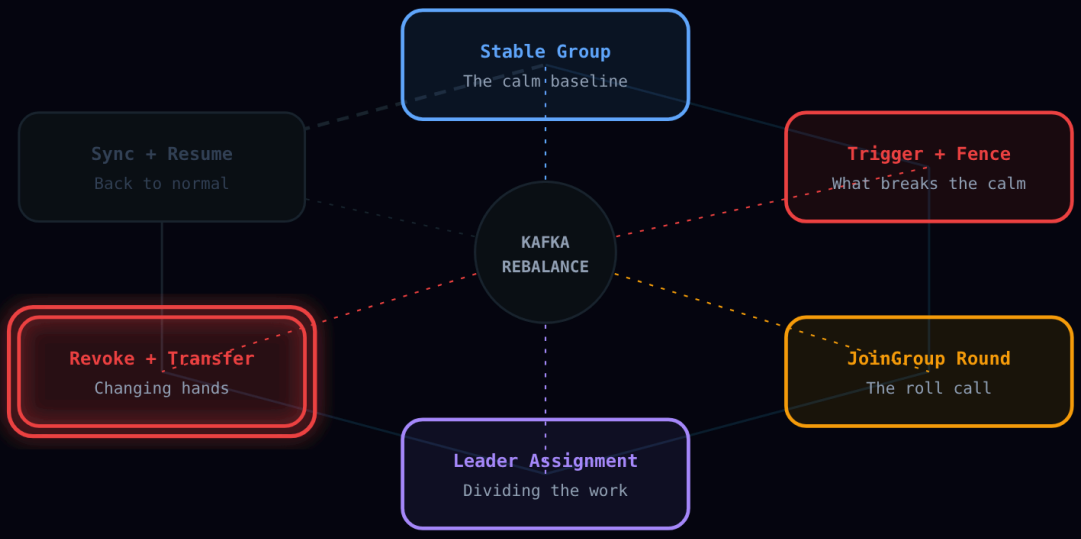
Heartbeats switch to the new generation. This is how the coordinator knows the rebalance has actually landed, not just been sent. Switch the Protocol control to see that in cooperative mode, some partitions never stopped serving.

Fetches resume from the new owners. In cooperative mode, only the moved partitions experienced any downtime. All other partitions kept flowing throughout the entire rebalance.

The group is stable again with a new live owner map and a new generation number. That brings us full circle. Now let us step back and see the whole picture together.

STAGE 8 OF 8

Recap



08 RECAP

We started by learning what a stable consumer group looks like. Every partition had an owner, every consumer was heartbeating, and the generation number said: this is the current truth.

Then we saw how one event can break that calm. A join, a leave, or a topic change fences the old generation and forces every consumer to stop and rejoin. The trigger is never subtle. It stops the whole group.

In the JoinGroup round, members gathered into one coherent rebalance attempt. The coordinator collected subscriptions and elected a leader for this specific round.

The leader computed a partition map using one of three assignor strategies. Range, RoundRobin, and Sticky each distribute work differently, and the choice has real consequences for how many partitions move.

The transfer phase was the operational heart of the rebalance. Eager mode revokes everything. Cooperative mode only revokes what must move. That one protocol choice determines how much throughput you lose during churn.

Finally, SyncGroup fanned the new map out, heartbeats switched generation, and polling resumed. The rebalance was complete only after every consumer had installed its assignment and was actively fetching.

Here is the unified picture. A Kafka rebalance is a coordinated shutdown and restart of partition ownership, triggered by change, organized through leader election and assignment, and completed only when every consumer is heartbeating under the new generation. Understanding each phase helps you tune timeouts, choose the right protocol, and diagnose slow rebalances in production.