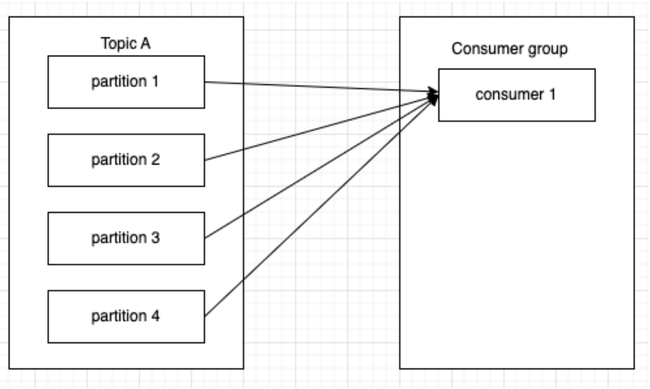
Kafka Consumers

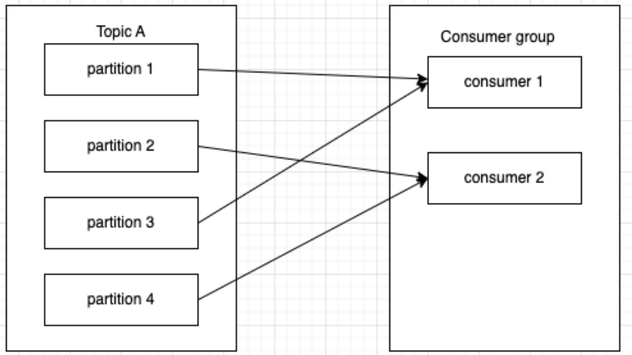
Overview, consumer configs, commits and offsets, rebalance listeners, standalone vs consumer groups

Sairam Elangovan

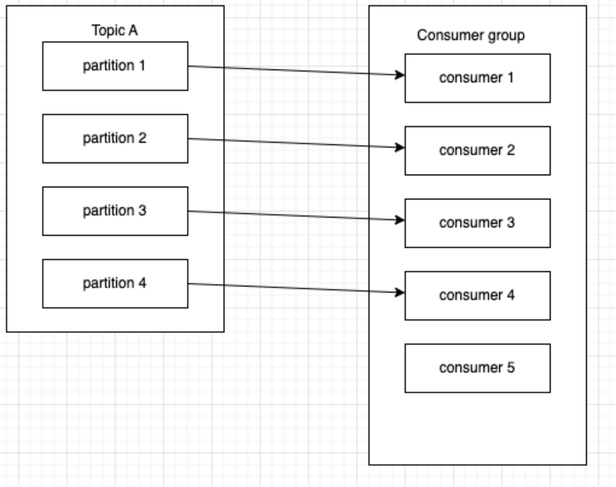
Consumers reading data from topic partitions



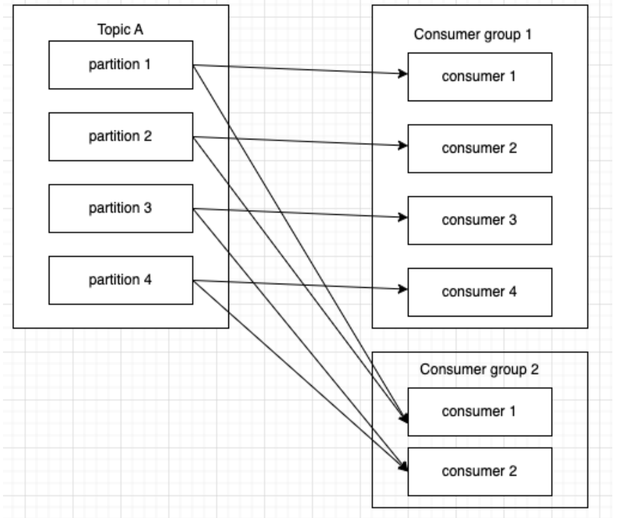
1 consumer consuming 4 partitions



2 consumers consuming mutually exclusive partitions



excess consumers in group compared to partitions



2 consumer groups consuming data independent of each other

Partition rebalancing in consumer groups

Moving partition ownership from one consumer to another is called a rebalance.

- Partition rebalancing in consumer groups occur when:
  - New consumer is added
  - Consumer is removed from group
  - Partitions are added in the topic

Types of rebalances

- **Eager rebalance**: all consumers stop consuming, give up their owner- ship of all partitions, rejoin the consumer group, and get a brand-new partition assignment. This is essentially a short window of unavailability of the entire consumer group. The length of the window depends on the size of the consumer group as well as on several configuration parameters.

- **Cooperative rebalances**(aka incremental rebalances): Cooperative rebalances typically involve reassigning only a small subset of the partitions from one consumer to another, and allowing consumers to continue processing records from all the partitions that are not reassigned.

- Initially, the consumer group leader informs all the consumers that they will lose ownership of a subset of their partitions, then the consumers stop consuming from these partitions and give up their ownership in them. In the second phase, the consumer group leader assigns these now orphaned partitions to their new owners. - **suitable for large consumer groups**

Static group membership

- By default, the identity of a consumer as a member of its consumer group is transient. When consumers leave a consumer group, the partitions that were assigned to the consumer are revoked, and when it rejoins, it is assigned a new member ID and a new set of partitions through the rebalance protocol.
- If you configure a consumer with a unique group.instance.id, which makes the consumer a *static* member of the group.
- If two consumers join the same group with the same group.instance.id, the second consumer will get an error saying that a consumer with this ID already exists.
- It is useful when re-creating this cache is time-consuming, you don’t want this process to happen every time a consumer restarts.
- It is important to note that static members of consumer groups do not leave the group proactively when they shut down, and detecting when they are “really gone” depends on the session.timeout.ms configuration. You’ll want to set it high enough to avoid triggering rebalances on a simple application restart but low enough to allow automatic reassignment of their partitions when there is more significant downtime, to avoid large gaps in processing these partitions.

Thread safety

- You can’t have multiple consumers that belong to the same group in one thread, and you can’t have multiple threads safely use the same consumer.
- One consumer per thread is the rule. To run multiple consumers in the same group in one application, you will need to run each in its own thread.
- It is useful to wrap the consumer logic in its own object and then use Java’s ExecutorService to start multiple threads, each with its own consumer.
- https://www.confluent.io/blog/tutorial-getting-started-with-the-new-apache-kafka-0-9-consumer-client/

Consumer configurations

- fetch.min.bytes
- fetch.max.wait.ms
- fetch.max.bytes
- client.id
- client.rack
- group.instance.id

Consumer configurations (contd)

- enable.auto.commit
- auto.offset.reset
- session.timeout.ms and heartbeat.interval.ms(by default heartbeat is set to 1/3 of session timeout value)
- max.poll.interval.ms (default value is 5 mins and triggered only during a deadlock)
- partition.assignment.strategy (Range, RoundRobin, Sticky, Cooperative sticky)

Commits and offsets