**Взаимосвязь настроек max.poll.records, max.poll.interval.ms**

Конфигурация max.poll.interval.ms, по умолчанию равная 5 минут, означает максимальную задержку между вызовами poll(). Она устанавливает верхнюю границу количества времени, в течение которого потребитель может бездействовать, прежде чем получить больше записей. Если метод poll() не вызывается до истечения этого тайм-аута, то потребитель считается отказавшим, и группа выполняет перебалансировку, чтобы переназначить разделы другому члену группы потребителей. Для потребителей, использующих ненулевой group.instance.id, которые достигли этого тайм-аута, разделы не будут переназначены немедленно. Вместо этого потребитель перестанет отправлять тактовые импульсы, а разделы будут переназначены по истечении session.timeout.ms. Это отражает поведение статического потребителя, который отключился.

Конфигурация max.poll.records – максимальное количество записей, которое может быть получено за один poll. По умолчанию равна 500.

Если на обработку сообщений, вытянутых через poll тратится больше, чем max.poll.interval.ms, то приложение-потребитель отправляет запрос на выход из группы потребителей координатору. При этом в логах Kafka регистрирует эти ошибки на уровне INFO, сообщая, что приложение-потребитель отправляет запрос на выход из группы координатору из-за истечения времени ожидания опроса потребителя.

Проблема может не показываться явно очень долго, пока размеры вытаскиваемых пачек маленькие. Но если произойдёт какой-то скачок нагрузки или накопится большая очередь, то может вылезти ещё и такая проблема, если сообщения обрабатываются медленно (медленнее 600 мс).

**О важности auto.offset.reset конфигурации**

За чтение сохраненных сообщений по порядку из топика Kafka приложением-потребителем отвечают смещения (offset), которые хранятся в топике \_\_consumer\_offsets. Этот топик создается автоматически после настройки Kafka-кластера и используется для хранения информации о подтвержденных смещениях для каждого топика по принципу раздел (partition) на группу потребителей (groupID). Каждый вызов API фиксации приводит к отправке брокеру запроса фиксации смещения. Используя синхронный API, потребитель блокируется до успешного ответа на этот запрос. Это ожидание может снизить общую пропускную способность, т.к. ожидая возврата, потребитель простаивает вместо обработки записей. Когда диспетчер смещения получает запрос на фиксацию смещения (OffsetCommitRequest), он добавляет запрос в топик \_\_consumer\_offsets. А менеджер смещения отправит потребителю ответ об успешной фиксации смещения только тогда, когда все реплики \_\_consumer\_offsets получат информацию об этом смещении.

Как только потребитель Kafka начинает получать данные из топика, он фиксирует смещение последнего полученного сообщения во внутреннем топике \_\_consumer\_offsets брокера. Так потребитель сможет определить смещение, с которого он должен начать чтение топика в следующем запросе. С этим связана конфигурация брокера offsets.retention.minutes для группы потребителей, значение которой по умолчанию задано 1440 минут, что составляет сутки (24 часа). Этот период означает время хранения смещений сообщений после того, как группа потребителей станет пустой. Для standalone потребителей после истечения срока времени последней фиксации плюс этого периода хранения зафиксированное потребителем смещение будет сброшено, и consumer уже не сможет найти его в топике \_\_consumer\_offsets. В случае отсутствия записи в топике \_\_consumer\_offsets поведение потребителя будет зависеть от конфигурации auto.offset.reset.

Итак, auto.offset.reset важен при:

1. Удалении информации об offset потребителя из kafka
2. При возникновении offset-out-of-range исключения в потребителе
3. При начале вычитки из топика
4. При создании партиций в уже существующем топике

Кейсы, когда можно потерять сообщения, если не указать auto.offset.reset = earliest (в kafka-clients по умолчанию latest):

1. Допустим существует топик, существуют producer и consumer приложения для этого топика. Теперь у топика увеличили кол-во партиций. Producer начинает записывать в эти новые партиции сообщения из-за roud-robing распределения при Null ключе, а consumer ещё не обновил свою метаинформацию об этом топике (Она обновляется либо по таймауту либо в некоторых других случаях). Соответственно в этих новых партициях копятся сообщения, а потом, когда consumer назначатся эти партиции без earliest настройки эти сообщения будут пропущены.
2. Если вручную сделать seek() на несуществующий offset, то consumer получит ошибку offset-out-of-range и offset для такого consumer будет назначен заново, на основании auto.offset.reset -> можно пропустить сообщения, если не earliest значение.
3. Если потушить под и запись об offset удалится, а сообщения в топик будут поступать. Т.е. даже если нет никаких ошибок.
4. Допустим накопится большой лаг у consumer. И на его обработку потребуется времени больше, чем retention у топика. -> Сообщения начнут пропадать раньше, чем они будут обработаны. В такой ситуации при очередном Poll у consumer возникнет ошибка Fetch position is out of range for partition… resetting offset. И в таком случае, если будет стоять auto.offset.reset = latest, то offset для этой партиции назначится самый последний, минуя весь лаг.

*Интересное дополнение:* когда я это тестировал я записывал в топик большое кол-во сообщений. Потом, между poll у consumer я удалял эти сообщения и записывал ещё много новых, имитируя исчезновение сообщений по retention (потому что настроить так, чтобы они автоматически удалились в нужный момент по retention было сложнее). И consumer по итогу получал position is out of range for partition и перескакивал все новые сообщения. Но интересное как раз в том, что делал он это не сразу. Я пробовал и с удалением по retention и с ручным clear topic в kafka ui. Результат один и тот же. Consumer ещё какое-то время продолжает вычитывать старые сообщения, которых нет в kafka-ui в топике. Даже если топик совсем пуст он продолжает читать вроде как удалённые сообщения. При этом вычитка ведётся с max.poll.records = 1. Т.е. kafka ещё какое-то время не удаляет реально сообщения, наверное, если есть живой потребитель, который потенциально может их вычитать. И сколько потребитель будет вычитывать несуществующие сообщения как будто зависит не от времени, т.к. я оставлял и на 40 минут и всё равно consumer медленно вычитывал по одному сообщению в минуту из пустого топика и за 40 минут не возникло ошибки, а дальше я ждать не стал. А зависит просто от кол-ва сообщений, после которых потребитель пробует обновить у себя информацию об Offset. Т.к. когда я сделал всё тоже самое, но сообщения обрабатывались быстро, то потребитель быстро вычитал несколько сотен несуществующих сообщений, потом у него возникло position is out of range for partition и потом он перепрыгнул новые сообщения, до которых так и не добрался. От чего зависит сколько несуществующих сообщений вычитает потребитель и сам механизм работы пока непонятен. Зато можно вызвать fetch position is out of range в случае описанном в 4 не только если по какой-то причине она сама произойдёт, но и если начнётся ребалансировка consumer (перезапуск пода, или пересоздание consumer и т.д.) и тогда весь лаг тоже будет пропущен