OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Транзакции

Непомнящий Евгений

Разработчик Java/Kotlin IT-Sense

@evgeniyN

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в TG

Задаем вопрос
в чат или **голосом**

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара

- Введение в проблему
- Idempotent Producer
- Транзакции

Цели вебинара

После занятия вы сможете

1. Рассмотреть транзакции
2. Изучить процесс работы с транзакциями

Смысл

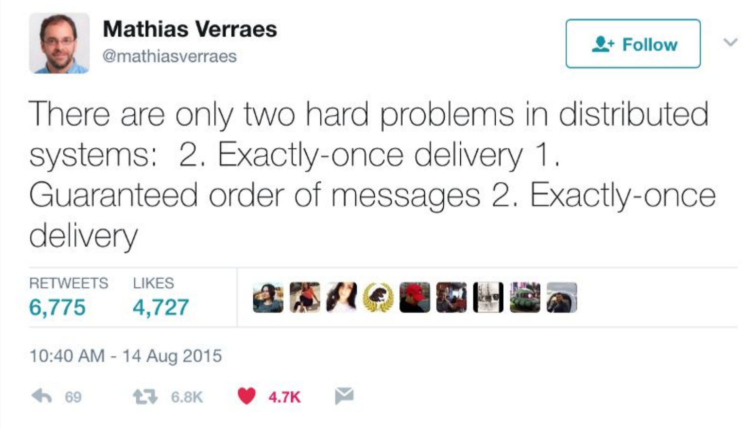
Зачем вам это уметь

1. Эти возможности позволяют в некоторых ситуациях получить гарантию Exactly Once практически бесплатно
2. Однако далеко не всегда, поэтому надо разобраться в тонкостях

Введение в проблему

Гарантии в Kafka

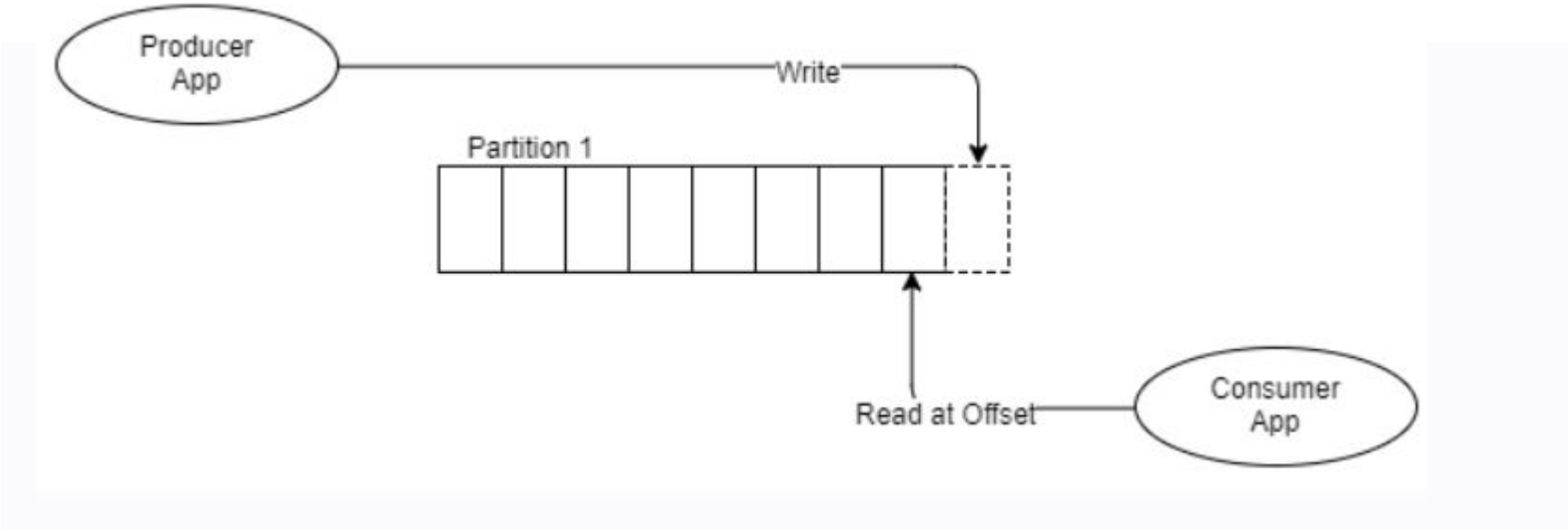
- at most once (нет гарантий доставки)
- at least once (сообщение будет доставлено, но возможны повторы)
- exactly* once (сообщение будет доставлено и ровно один раз)



* много мелкого текста.

Где могут быть проблемы?

- producer несколько раз записал одно и тоже сообщение
- consumer несколько раз прочитал одно и тоже сообщение



Producer

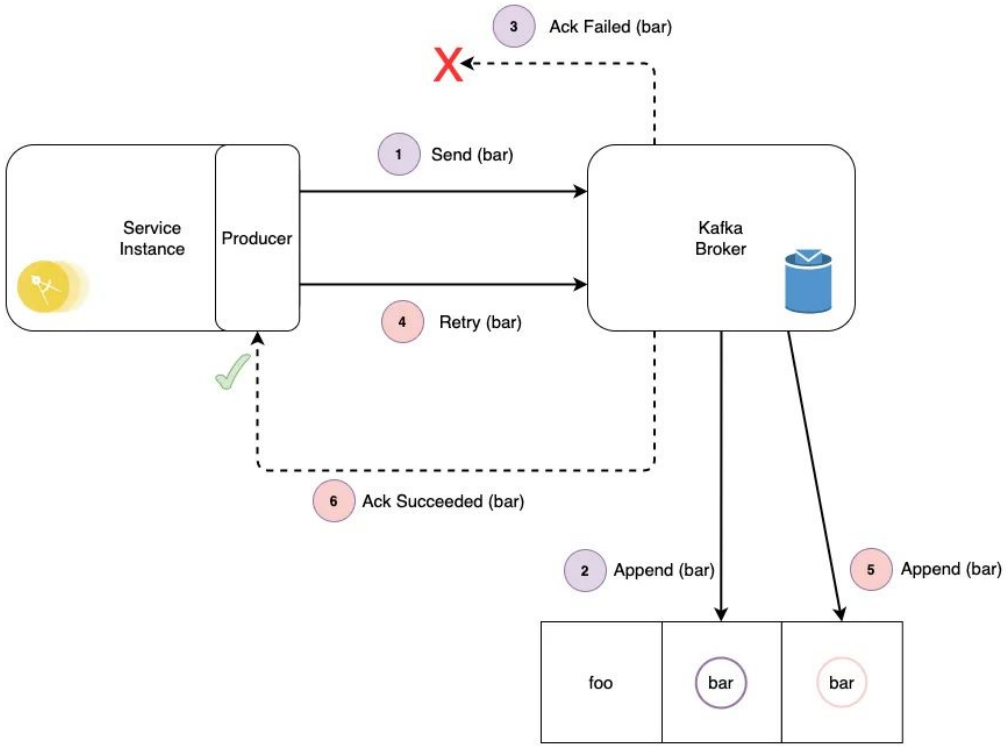
1. Что-то происходит (входящий запрос или сообщение, бизнес-процесс перешел на очередной шаг, …)
   a. producer успешно обрабатывает эту ситуацию
   b. producer падает
2. Producer пишет сообщение в kafka
   a. kafka получила сообщение и сохранила в нужное число реплик
   b. kafka не получила сообщение или не смогла сохранить
3. Producer получает ack
   a. ack доходит
   b. ack не доходит и повторяет библиотека
   c. ack не доходит и логика producer получает ошибку
4. Producer “отмечает”, что сообщение отправлено
   a. отвечает на входящий запрос
   b. подтверждает входящее сообщение
   c. пишет что-то в свою БД

Consumer

1. Получает пачку сообщений
2. Берет очередное сообщение из полученных
3. Обрабатывает его
    a. Отправляет сообщения в kafka в другой топик
    b. Отправляет запрос в другой сервис
    c. Меняет что-то в своей БД
4. Сообщает kafka о новом offset

Idempotent Producer

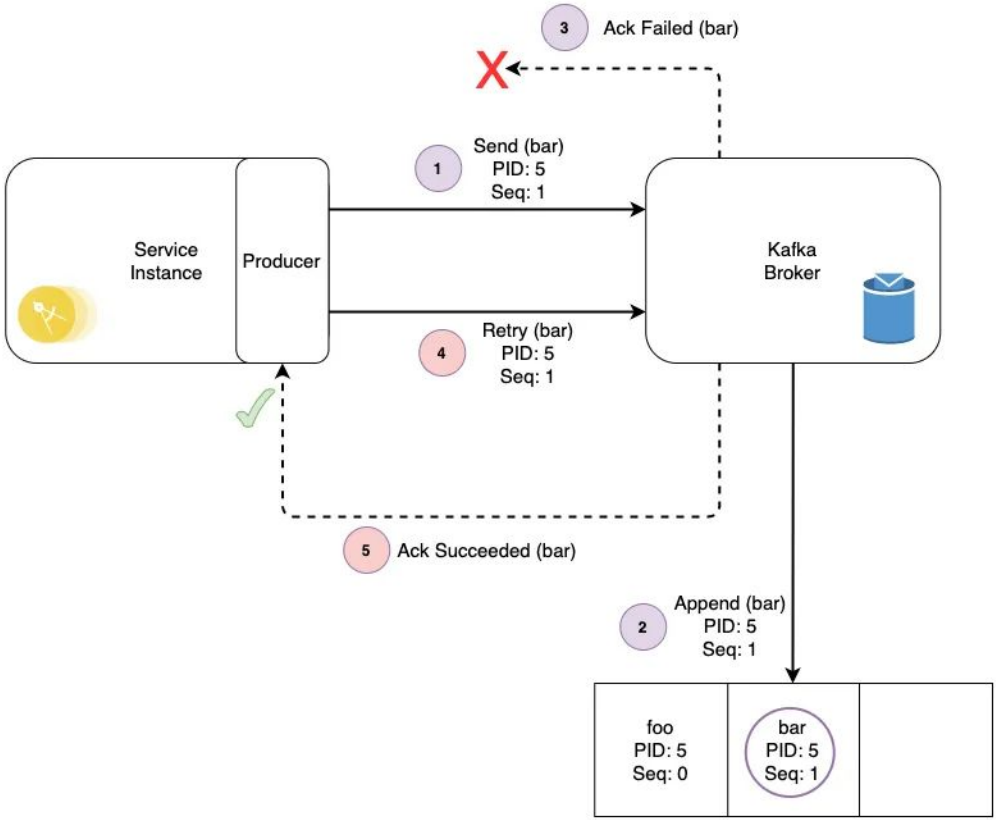
Какую проблему решаем



- Что-то происходит (входящий запрос или сообщение, бизнес-процесс перешел на очередной шаг, …)
- Producer пишет сообщение в kafka
- Kafka отправляет ack
  a. ack доходит
  b. ack не доходит и повторяет библиотека
  c. ack не доходит и логика producer получает ошибку
- Producer “отмечает”, что сообщение отправлено

p1….Ex1Problem

Решение



- `enable.idempotence=true`
- PID - идентификатор producer
- Seq - номер сообщения в рамках эпохи
- Накладные расходы малы

p1….Ex2Idempotent

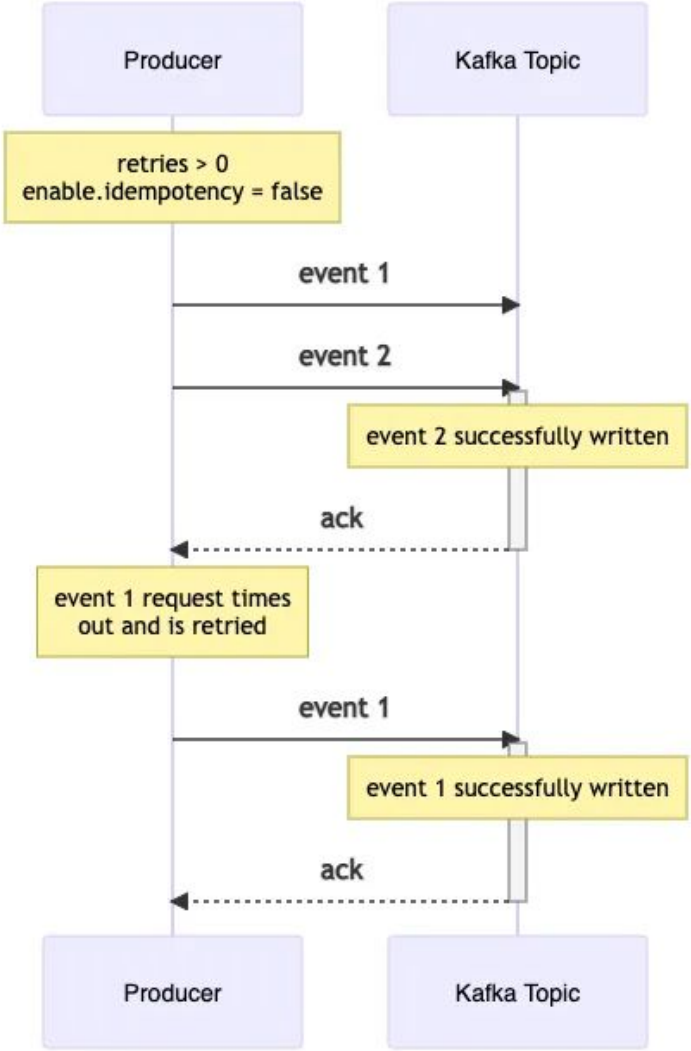
Параметры

1. Не настраивать `enable.idempotence`
   a. он будет включен автоматически (начиная с 3.0.0), если
      i. `max.in.flight.requests.per.connection` <= 5
      ii. `retries` > 0
      iii. `acks` = all
   b. иначе будет выключен
2. `enable.idempotence` = true - условия выше должны соблюдаться, иначе будет исключение при создании KafkaProducer

p1….Ex3InvalidConfig

Порядок сообщений при ошибках

Если идемпотентность отключена - защиты нет.
Если включена - кафка гарантирует порядок в такой ситуации



А если send вернул ошибку?

1. Что-то происходит (входящий запрос или сообщение, бизнес-процесс перешел на очередной шаг, …)
2. Producer пишет сообщение в kafka
3. Kafka отправляет ack
   a. ack доходит
   b. ack не доходит и повторяет библиотека
   c. ack не доходит и логика producer получает ошибку
4. Producer “отмечает”, что сообщение отправлено

То все плохо - никакой защиты тут нет. Idempotent Producer работает только для повторов библиотеки. От повторов на уровне логики приложения он никак не защищает.

p1….Ex4ProducerError

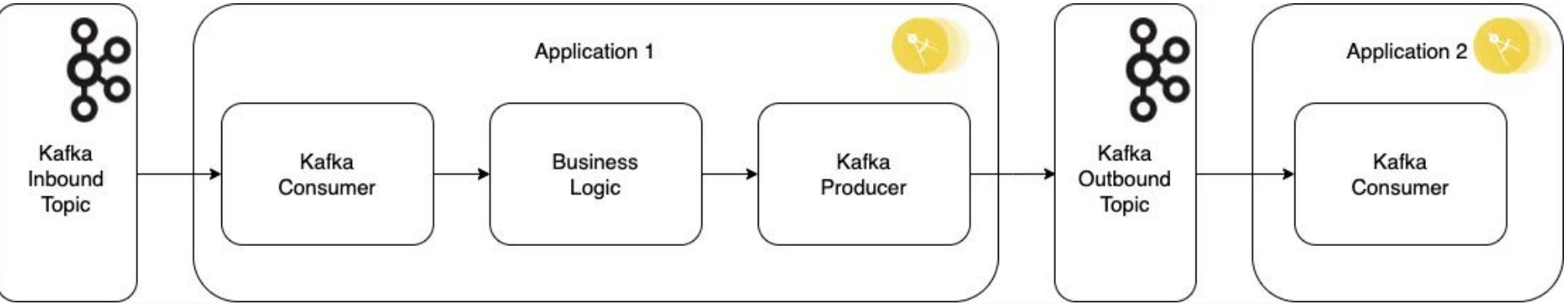
Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Exactly once

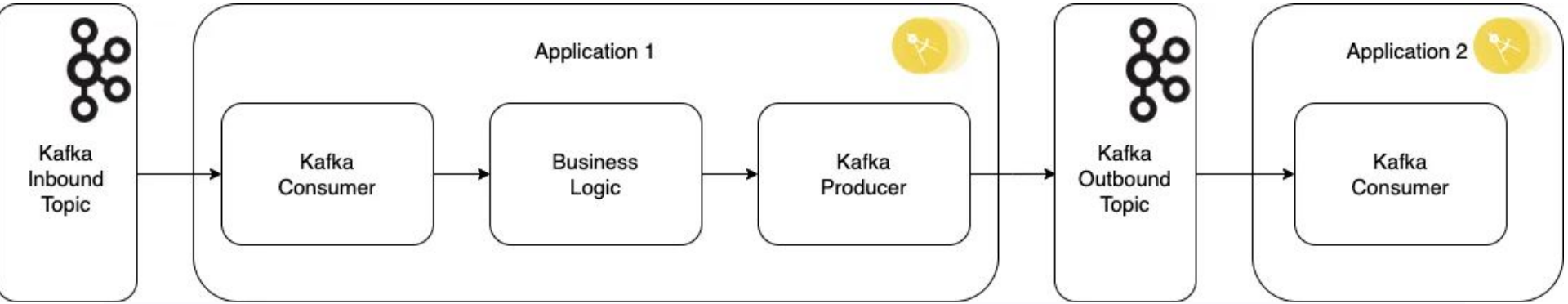
Какую проблему мы решаем



Сообщение из Inbound обрабатывается и приводит к отправке сообщения в Outbound.
Application 1 может быть запущена в нескольких экземплярах.
Application 1 может быть stateless и не иметь БД.

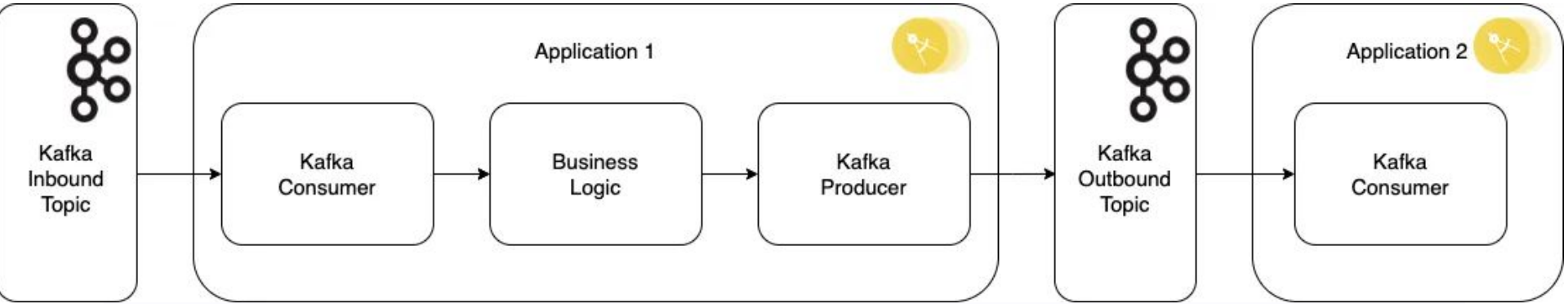
Надо, чтобы каждое сообщение из Inbound было обработано и сообщение в Outbound были отправлено ровно один раз и в том порядке, в котором читались сообщения из Inbound.

Как работает App1



1. Consumer получает пачку сообщений
2. Каждое сообщение
   a. обрабатывается бизнес-логикой
   b. Producer пишет сообщение в Outbound
   c. После успешной записи коммитится offset в Inbound

Что может пойти не так



1. App1/Producer отправил и задублировал сообщение - *решается Idempotent Producer*
2. Сообщение ушло в outbound, но App1 упал и не отправил ack - *решается транзакциями*
3. Зомби - см следующий слайд

Зомби

Инстанс 1	Инстанс 2	Кафка	Целевой топик
Получает сообщения 1, 2			
Обрабатывает 1, посылает A, отправляет offset			A
Начинает долгую сборку мусора		Теряет инстанс 1, перебалансировка	
	Получает сообщение 2		
Просыпается, обрабатывает 2, посылает B	Обрабатывает 2, посылает B		A B B

Транзакции

Транзакция позволяет записать сообщения в один или несколько топиков или все успешно или ни одно не запишется*.
offset хранятся в топике, их коммит суть запись в топик.
Т.е. в рамках транзакции можно отправить сообщения в целевой топик и закоммитить offset как единую атомарную операцию - или успешную или нет.

p2….Ex5Transaction
p2….Ex6ReadWrite

* на самом деле запишется, но специальным образом настроенный consumer его не увидит

Транзакции

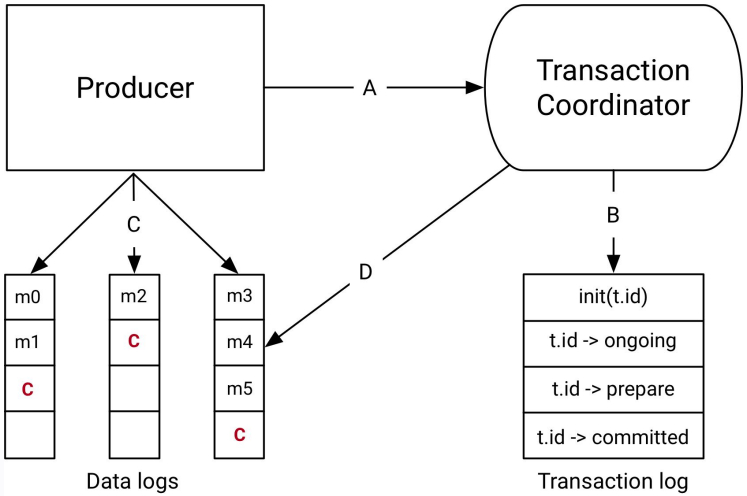
- `ProducerConfig.TRANSACTIONAL_ID_CONFIG` должен быть задан, это включает поддержку транзакций, а также требует идемпотентности producer
- `producer.initTransactions()` должен быть вызван однократно перед началом работы
- `producer.beginTransaction()` начинает транзакцию
- `producer.send()` идет в рамках транзакции, если она была начата
- `producer.sendOffsetsToTransaction()` отправляет offset в рамках транзакции
- `producer.commitTransaction() / producer.abortTransaction()` - блокирует поток до отправки всех сообщений, завершает или отменяет транзакцию

isolation.level

- Транзакции влияют только на тех потребителей, кто настроил `isolation.level = read_committed`
  - **По умолчанию это не так!**
- для таких потребителей сообщения из партиции не будут приходить, пока транзакция не будет завершена (успешно или нет)
  - В том числе сообщения от нетранзакционных продьюсеров

p2….Ex7IsolationLevel

Как устроены транзакции



A. Producer -> Coordinator
    a. P регистрирует свой transaction.id
    b. TC абортит активную транзакцию с таким id
    c. TC “огораживает” P с таким id (защита от зомби)
    d. Когда P первый раз посылает данные в партицию, он сначала уведомляет об этом TC
    e. abort / commit отправляется в TC
B. Coordinator -> Transaction log
    a. действия из A регистрируются в TL (служебный топик)
C. Producer -> Topic / Partition
D. Coordinator -> Topic / Partition (после commit/abort)
    a. Обновляет состояние транзакции, после этого транзакция гарантированно завершается
    b. Пишет маркеры коммита во все задействованные партиции
    c. Помечает транзакцию завершенной

Защита от зомби

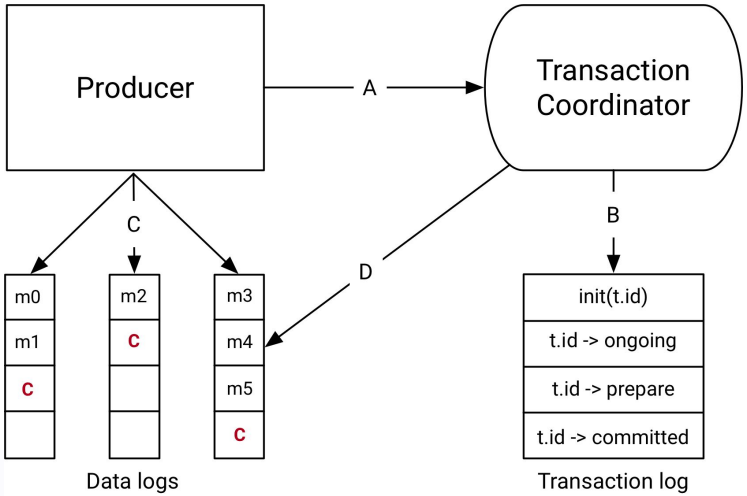
Инстанс 1	Инстанс 2	Кафка	Целевой топик
initTransaction (t.id = X)			
Получает сообщения 1, 2			
Обрабатывает 1, посылает A			A
Начинает долгую сборку мусора		Теряет инстанс 1, перебалансировка	
	initTransaction (t.id = X)	эпоха увеличена	
	Получает сообщение 2		
Просыпается, обрабатывает 2, посылает B		Сообщает об ошибке, так как эпоха старая	A
	Обрабатывает 2, посылает B		A B

Защита от зомби

Producer -> Coordinator

p2….Ex8Fenced

Накладные расходы



- Состоят из
  - P -> TC
  - TC -> TL
  - TC -> Topic/Partitions
- Не зависят от объема транзакции
- Чем больше сообщений в транзакции, тем меньше (относительно) накладные расходы
- Однако большие транзакции вызывают блокировку партиции для read_committed потребителей
- `transaction.timeout.ms` - момент принудительной отмены транзакции (по умолчанию 60 с)

AdminClient

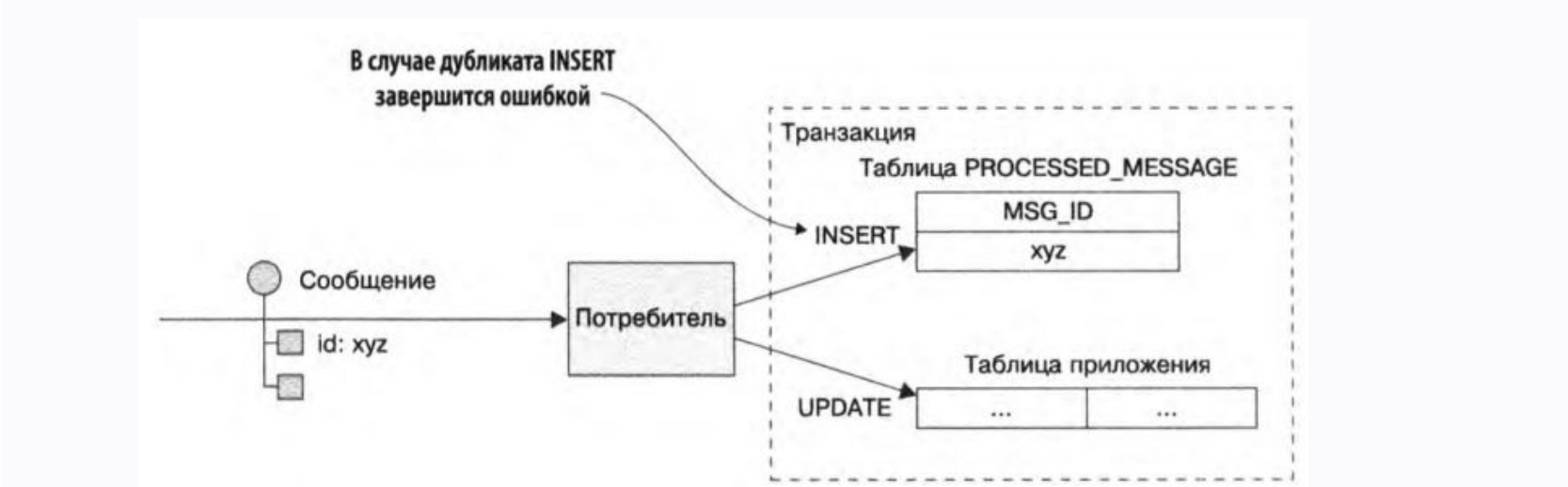
```
AbortTransactionResult abortTransaction(AbortTransactionSpec spec)
AbortTransactionSpec(
  TopicPartition topicPartition,
  long producerId,
  short producerEpoch,
  int coordinatorEpoch

ListTransactionsResult listTransactions()
TransactionListing(
  String transactionalId,
  long producerId,
  TransactionState transactionState

DescribeTransactionsResult describeTransactions(Collection<String> transactionalIds)
```

Основное ограничение Exactly Once

Эта механика работает в случае “Читаем из Kafka, Меняем, Пишем в Kafka”.
Если появляется сторонняя система (БД, сервис и т.п.) - вы должны полагаться только на at least once и сами дедуплицировать сообщения



Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Ключевые тезисы

1. Idempotent Producer - классная и почти бесплатная фича, включенная по умолчанию
2. Idempotent Producer не сработает в случае, если продюсер сам сделал повтор (если повторы на уровне библиотеки не помогли)
3. Транзакции - возможность сделать множественную запись в топики атомарной
4. Транзакции надо явно включать как на стороне продьюсера (transaction.id), так и на стороне потребителя (isolation.level)
5. Транзакции приводят к блокировке партиции для потребителей с транзакциями
6. Idempotent Producer + Транзакции = Exactly Once для Прочитать+Обработать(без БД)+Записать

Что почитать

- https://www.infoq.com/articles/no-reliable-messaging/
- https://medium.com/lydtech-consulting/kafka-transactions-part-1-exactly-once-messaging-9949350281ff
- https://medium.com/lydtech-consulting/kafka-transactions-part-2-spring-boot-demo-ce066713c7a7
- https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/
- https://developer.confluent.io/learn/kafka-transactions-and-guarantees/
- https://cwiki.apache.org/confluence/display/KAFKA/KIP-98+-+Exactly+Once+Delivery+and+Transactional+Messaging
- https://www.youtube.com/watch?v=PgkRhlUwYyE

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Непомнящий Евгений

Разработчик Java/ Kotlin IT-Sense

@evgeniyN