OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Kafka Consumer API

Чащина Александра

Big Data Engineer

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в Телеграм

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

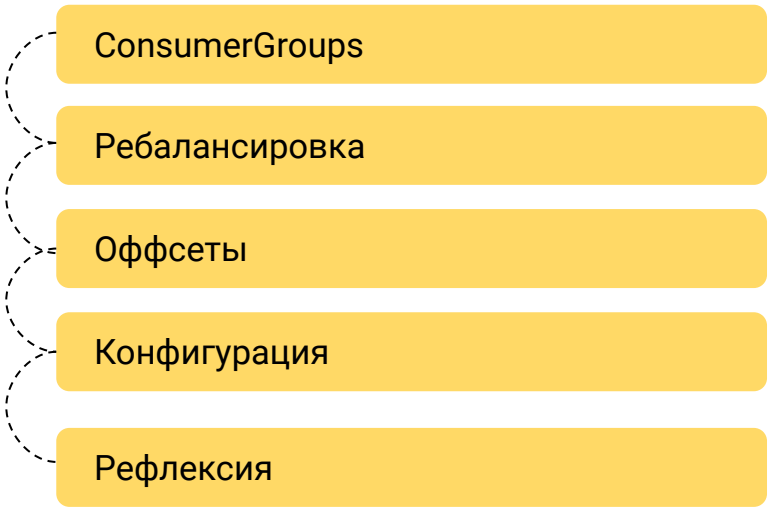
Документ

Ответьте себе или
задайте вопрос

Карта курса


Введение
Развертывание кластера Apache Kafka
Программирование
Потоковая обработка данных
Интеграция
Проектная работа

Маршрут вебинара



Цели вебинара

После занятия вы сможете

1. Объяснить, что такое ConsumerGroups и зачем они нужны
2. Использовать Consumer API для отслеживания оффсетов
3. Настраивать Consumer

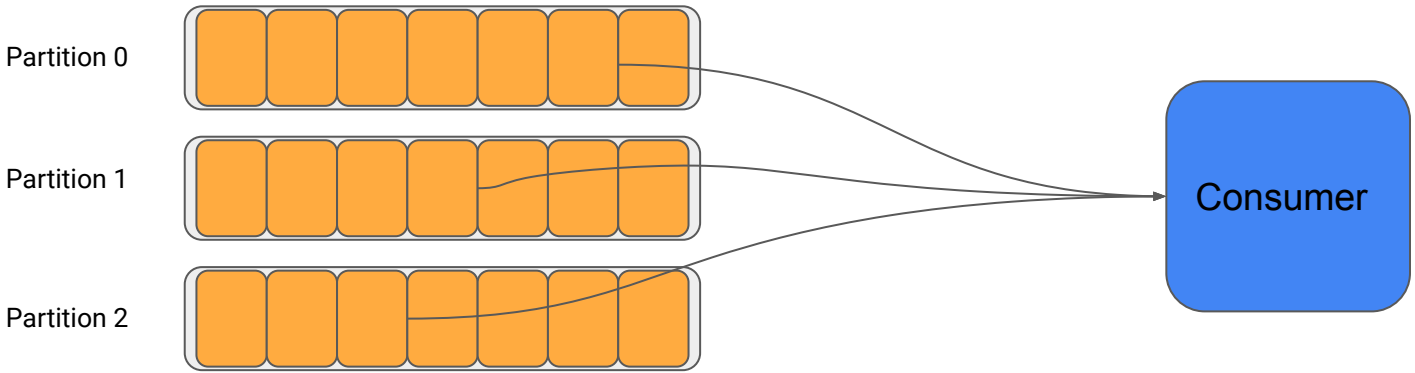
Смысл

Зачем вам это уметь

1. Механика чтения сообщений требует особого внимания
2. Знание механики позволяет тонко настраивать приложение, читающее сообщения Kafka

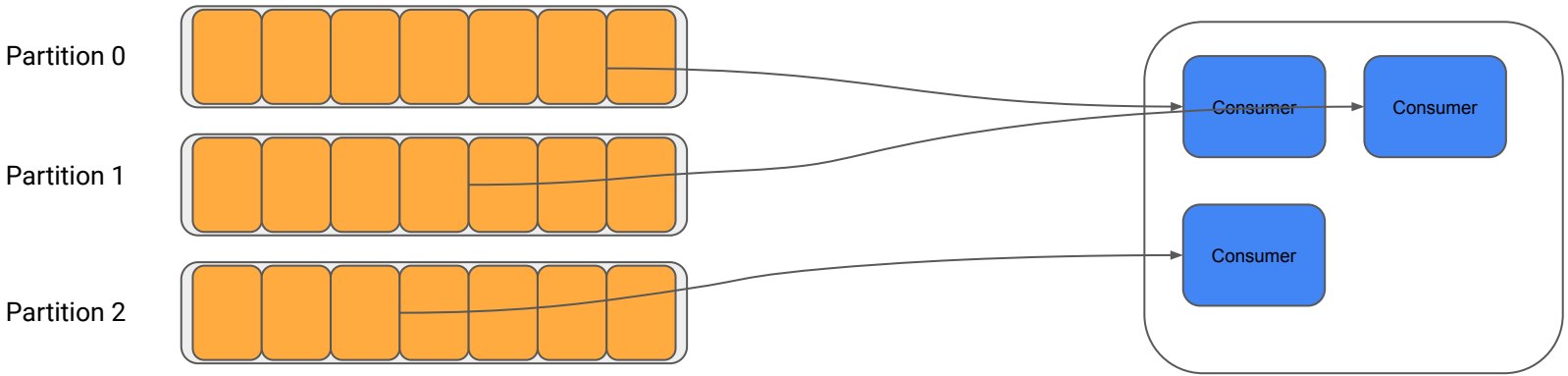
ConsumerGroups

Чтение топика одним процессом Consumer



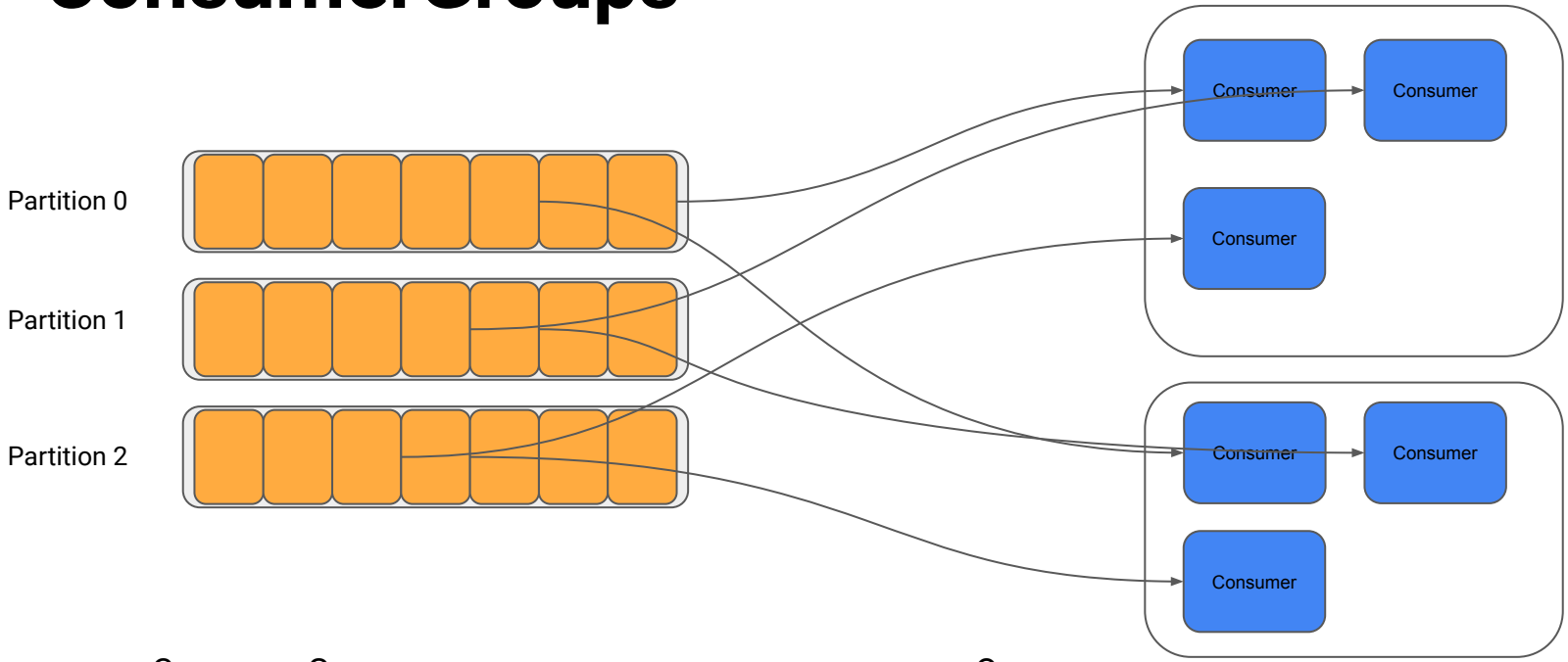
Один Consumer читает сообщения из всех 4 партиций одновременно.

Чтение топика группой Consumer (ConsumerGroup)



3 Consumer формируют ConsumerGroup. В группе чтение из партиций распределяется автоматически. Одна партиция читается полностью одним Consumer. Один Consumer может читать несколько партиций. Принадлежность к группе определяется `group.id`.

Чтение топика несколькими группами ConsumerGroups



Несколько ConsumerGroups могут читать один и тот же топик. Одна группа может читать несколько топиков.

Распределение нагрузки в ConsumerGroups

Настраивается конфигом `partition.assignment.strategy`. Есть 4 стратегии распределения:

Распределение нагрузки в ConsumerGroups

Настраивается конфигом `partition.assignment.strategy`. Есть 4 стратегии распределения:

1. `RangeAssignor` (по умолчанию)

Распределение нагрузки в ConsumerGroups

Настраивается конфигом `partition.assignment.strategy`. Есть 4 стратегии распределения:

1. `RangeAssignor` (по умолчанию)
2. `RoundRobin`

Распределение нагрузки в ConsumerGroups

Настраивается конфигом `partition.assignment.strategy`. Есть 4 стратегии распределения:

1. `RangeAssignor` (по умолчанию)
2. `RoundRobin`
3. `StickyAssignor`

Распределение нагрузки в ConsumerGroups

Настраивается конфигом `partition.assignment.strategy`. Есть 4 стратегии распределения:

1. `RangeAssignor` (по умолчанию)
2. `RoundRobin`
3. `StickyAssignor`
4. `CooperativeStickyAssignor`

Распределение нагрузки в ConsumerGroups

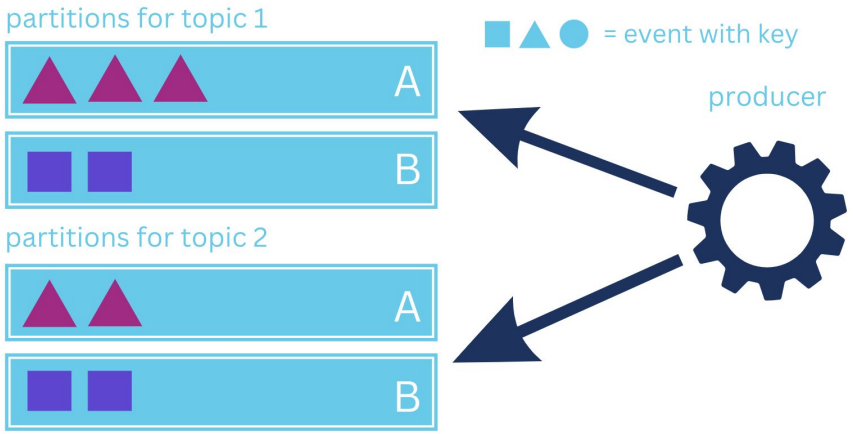
Настраивается конфигом `partition.assignment.strategy`. Есть 4 стратегии распределения:

1. `RangeAssignor` (по умолчанию)
2. `RoundRobin`
3. `StickyAssignor`
4. `CooperativeStickyAssignor`

Имеет смысл менять стратегии только тогда, когда одна группа читает несколько топиков.

Стратегия RangeAssignor

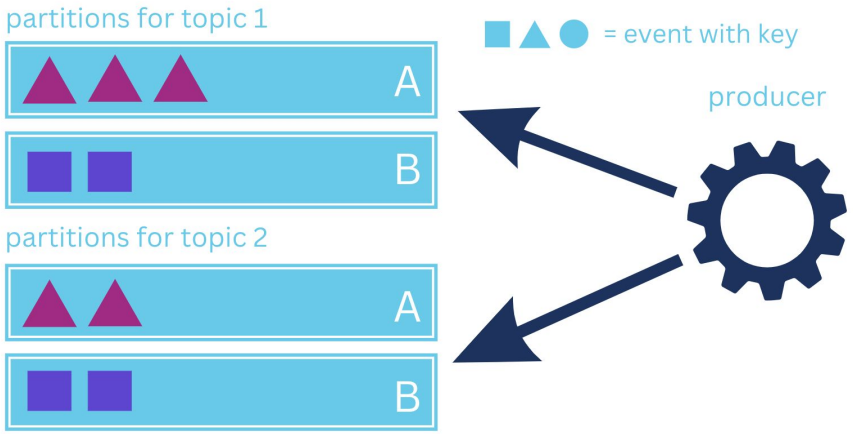
Топики должны быть co-partitioned:



Стратегия RangeAssignor

Топики должны быть co-partitioned:

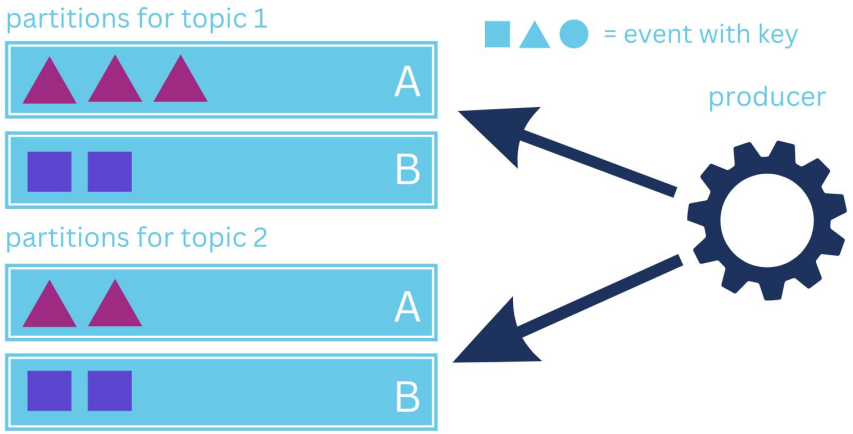
- Иметь одинаковое количество партиций;



Стратегия RangeAssignor

Топики должны быть co-partitioned:

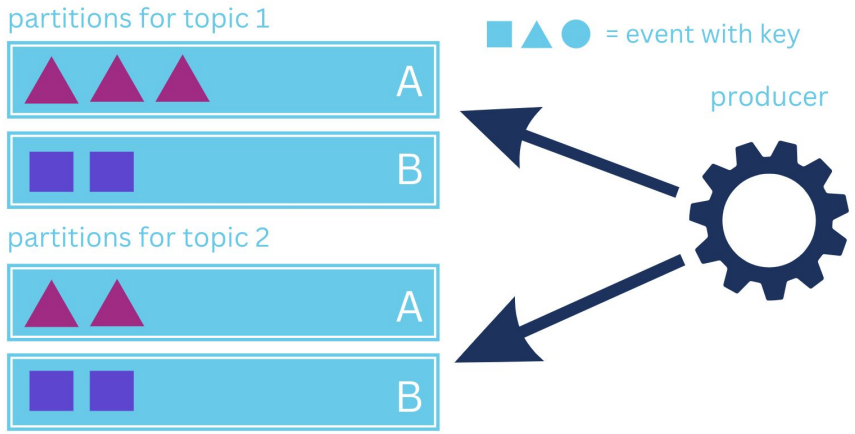
- Иметь одинаковое количество партиций;
- Иметь одинаковую стратегию партицирования;



Стратегия RangeAssignor

Топики должны быть co-partitioned:

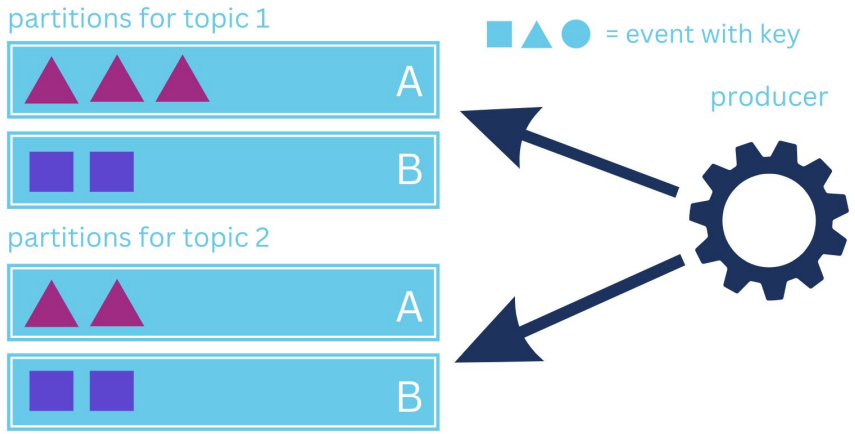
- Иметь одинаковое количество партиций;
- Иметь одинаковую стратегию партицирования;
- Быть партицированы по одним и тем же ключам.



Стратегия RangeAssignor

Топики должны быть co-partitioned:

- Иметь одинаковое количество партиций;
- Иметь одинаковую стратегию партицирования;
- Быть партицированы по одним и тем же ключам.

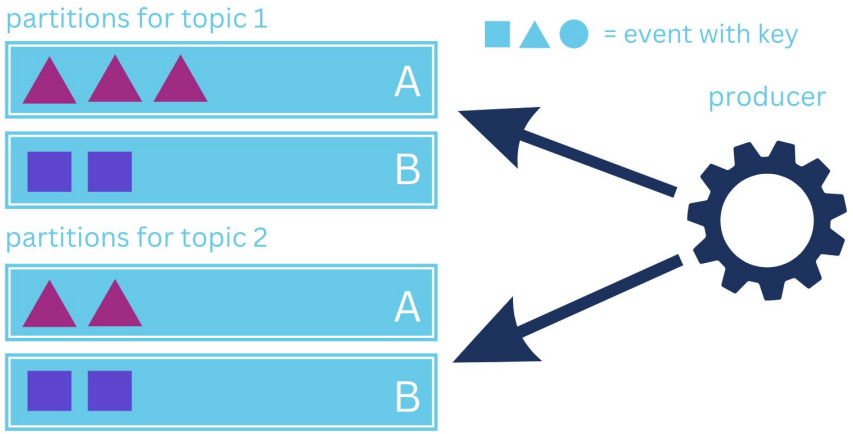


`RangeAssignor` используется тогда, когда необходимо в верном порядке читать сообщения из нескольких топиков. Эта стратегия гарантирует, что данные одного и того же ключа из двух топиков будут прочитаны одним Consumer.

Стратегия RangeAssignor

Топики должны быть co-partitioned:

- Иметь одинаковое количество партиций;
- Иметь одинаковую стратегию партицирования;
- Быть партицированы по одним и тем же ключам.



`RangeAssignor` используется тогда, когда необходимо в верном порядке читать сообщения из нескольких топиков. Эта стратегия гарантирует, что данные одного и того же ключа из двух топиков будут прочитаны одним Consumer.

Полезно, когда нужно соединить данные из нескольких топиков.

Другие стратегии

- `RoundRobin` - честное распределение партиций между процессами в ConsumerGroup. Используется тогда, когда важна максимальная скорость чтения.

Другие стратегии

- `RoundRobin` - честное распределение партиций между процессами в ConsumerGroup. Используется тогда, когда важна максимальная скорость чтения.
- `StickyAssignor` - сохранение существующего распределения партиций между процессами в ConsumerGroup в случае ребалансировки. Сокращает время overhead-операций. Может быть полезно для stateful-приложений.

Другие стратегии

- `RoundRobin` - честное распределение партиций между процессами в ConsumerGroup. Используется тогда, когда важна максимальная скорость чтения.
- `StickyAssignor` - сохранение существующего распределения партиций между процессами в ConsumerGroup в случае ребалансировки. Сокращает время overhead-операций. Может быть полезно для stateful-приложений.
- `CooperativeStickyAssignor` - то же самое, что и `StickyAssignor`, но стремится не блокировать процессы в случае балансировки.

Вопросы?

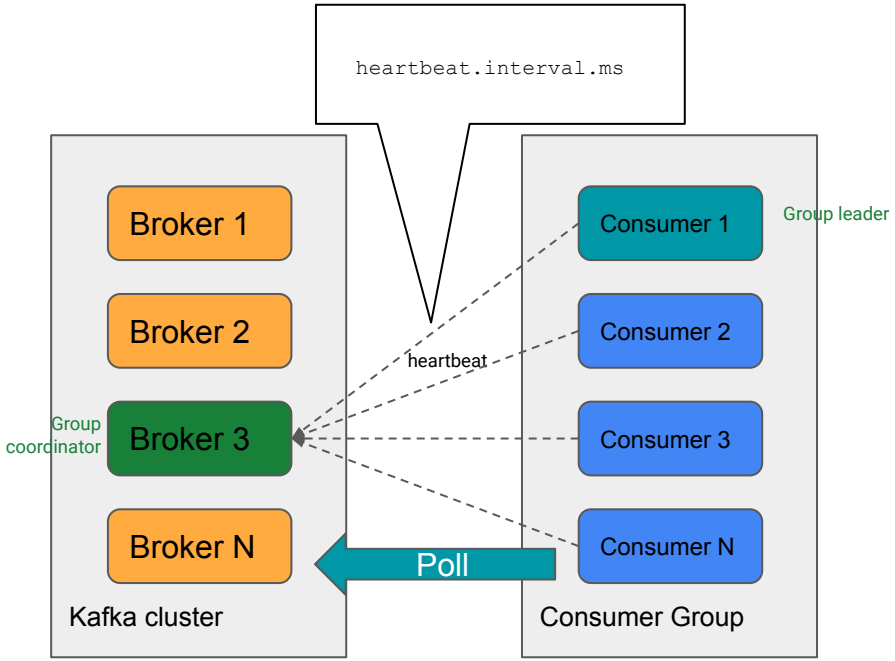
Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Ребалансировка

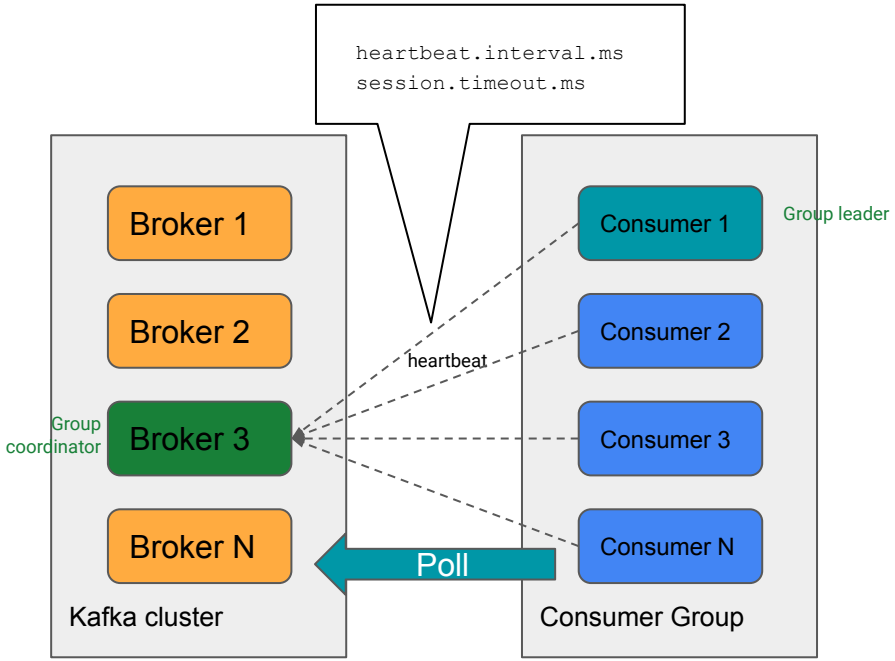
Отслеживание процессов Consumer

1. `heartbeat.interval.ms` - каждый Consumer отправляет сигнал координатору группы;



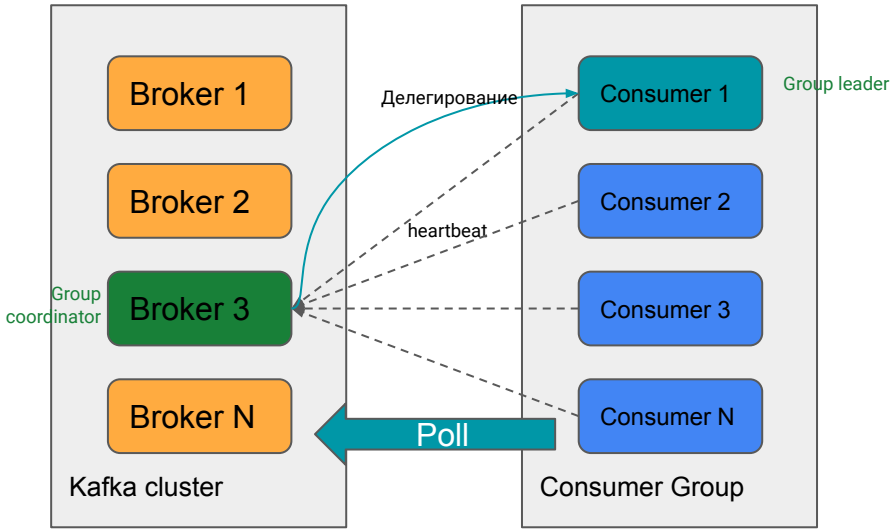
Отслеживание процессов Consumer

1. `heartbeat.interval.ms` - каждый Consumer отправляет сигнал координатору группы;
2. Если сигнал не получен в течение `session.timeout.ms` , то координатор приступает к ребалансировке;



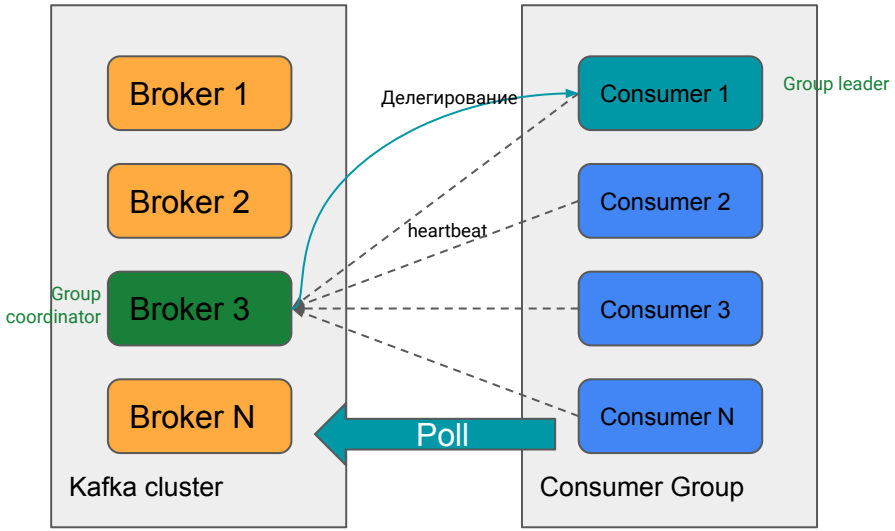
Отслеживание процессов Consumer

1. `heartbeat.interval.ms` - каждый Consumer отправляет сигнал координатору группы;
2. Если сигнал не получен в течение `session.timeout.ms` , то координатор приступает к ребалансировке;
3. Распределение партиций делегируется лидеру группы;



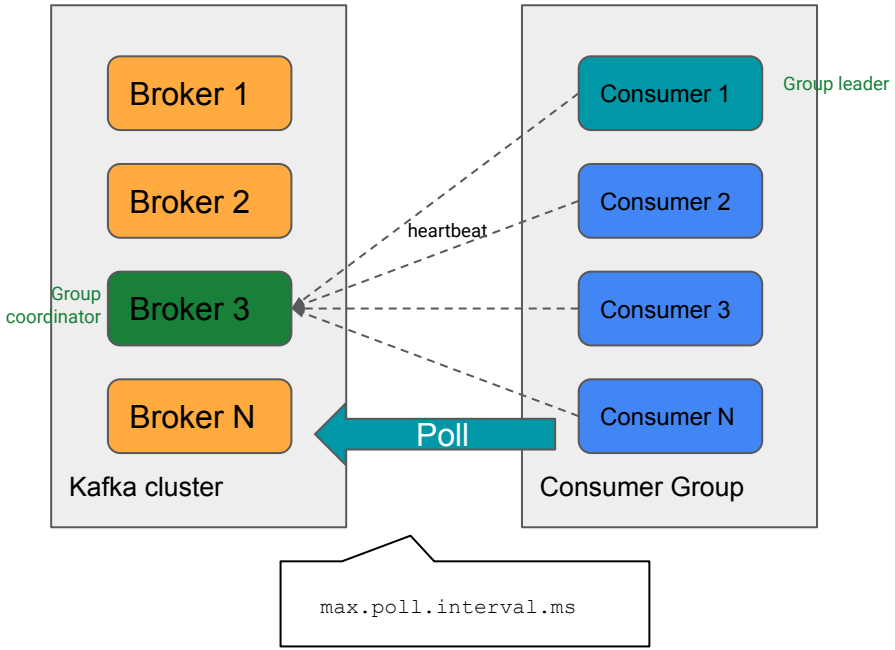
Отслеживание процессов Consumer

1. `heartbeat.interval.ms` - каждый Consumer отправляет сигнал координатору группы;
2. Если сигнал не получен в течение `session.timeout.ms`, то координатор приступает к ребалансировке;
3. Распределение партиций делегируется лидеру группы;
4. Координатор группы сообщает всем остальным Consumer их новые партиции;



Отслеживание процессов Consumer

1. `heartbeat.interval.ms` - каждый Consumer отправляет сигнал координатору группы;
2. Если сигнал не получен в течение `session.timeout.ms` , то координатор приступает к ребалансировке;
3. Распределение партиций делегируется лидеру группы;
4. Координатор группы сообщает всем остальным Consumer их новые партиции;
5. `max.poll.interval.ms` - интервал, за который один Consumer должен запросить новую порцию сообщений.



Rebalancing

Возникает, когда:

Rebalancing

Возникает, когда:

- Consumer покидает или присоединяется к группе;

Rebalancing

Возникает, когда:

- Consumer покидает или присоединяется к группе;
- Consumer меняет топик;

Rebalancing

Возникает, когда:

- Consumer покидает или присоединяется к группе;
- Consumer меняет топик;
- Меняется количество партиций топика.

Rebalancing

Возникает, когда:

- Consumer покидает или присоединяется к группе;
- Consumer меняет топик;
- Меняется количество партиций топика.

Не возникает при вызове `Consumer.pause()`

Rebalancing

Устойчивость к ошибкам
отдельных Consumer

Rebalancing

Устойчивость к ошибкам отдельных Consumer

Автоматическое масштабирование

Rebalancing

Устойчивость к ошибкам отдельных Consumer

Автоматическое масштабирование

Процесс чтения приостанавливается

Rebalancing

Устойчивость к ошибкам отдельных Consumer

Автоматическое масштабирование

Процесс чтения приостанавливается

Stateful-процессы должны заново вычислить состояние для новых партиций

Как избежать лишних ребалансировок

- Установить `heartbeat.interval.ms = ⅓ session.timeout.ms:`

Как избежать лишних ребалансировок

- Установить `heartbeat.interval.ms = ⅓ session.timeout.ms:`
  - Больше времени, чтобы выпавший Consumer подключился заново;
  - Больше времени на обнаружение серьезных поломок.

Как избежать лишних ребалансировок

- Установить `heartbeat.interval.ms = ⅓ session.timeout.ms:`
  - Больше времени, чтобы выпавший Consumer подключился заново;
  - Больше времени на обнаружение серьезных поломок.
- Установить достаточно времени для `max.poll.interval.ms`

Как избежать лишних ребалансировок

- Установить `heartbeat.interval.ms = ⅓ session.timeout.ms:`
  - Больше времени, чтобы выпавший Consumer подключился заново;
  - Больше времени на обнаружение серьезных поломок.
- Установить достаточно времени для `max.poll.interval.ms`
- Распределить Consumer по группам статически: `group.instance.ms`

Вопросы?

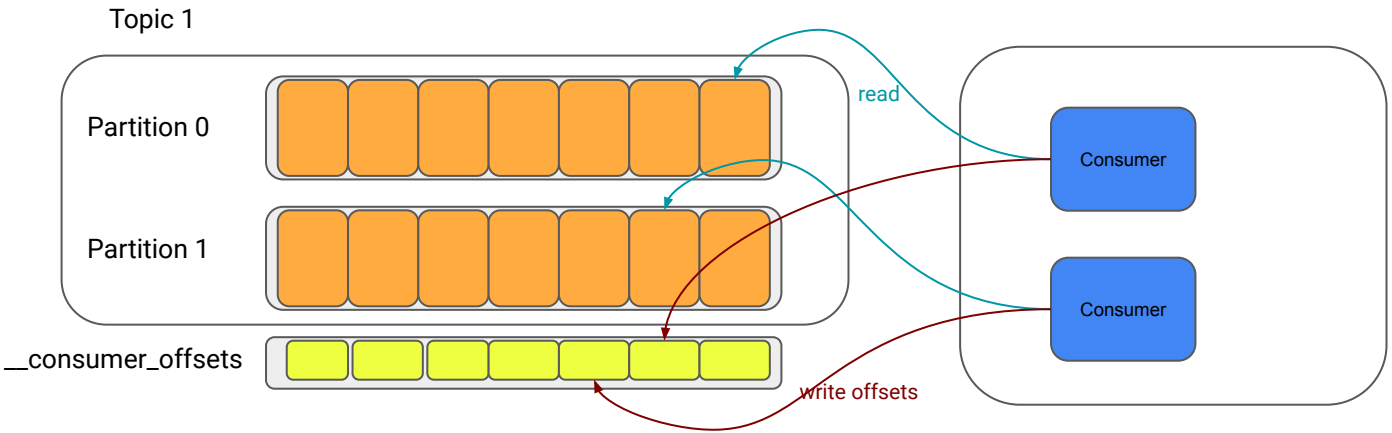
Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Оффсеты

Offsets

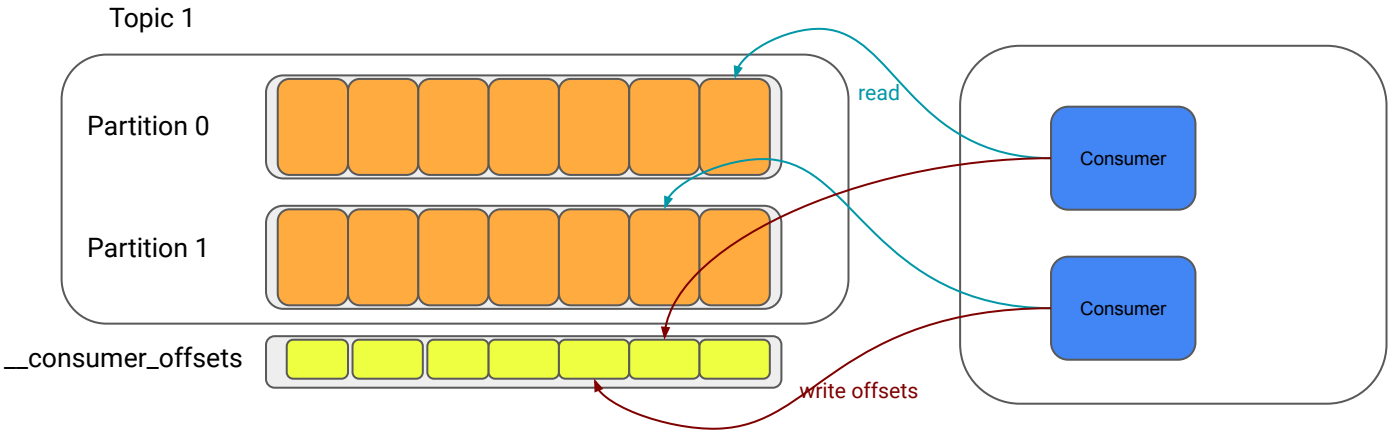
Offset - это порядковый номер сообщения в партиции.



Offsets

Offset - это порядковый номер сообщения в партиции.

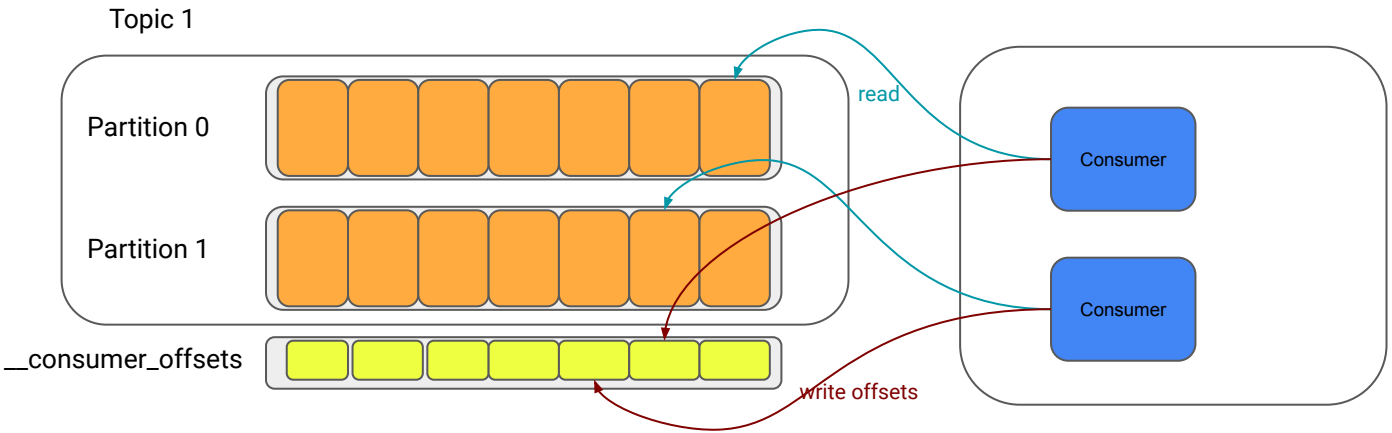
- Брокер отслеживает прогресс каждого Consumer;



Offsets

Offset - это порядковый номер сообщения в партиции.

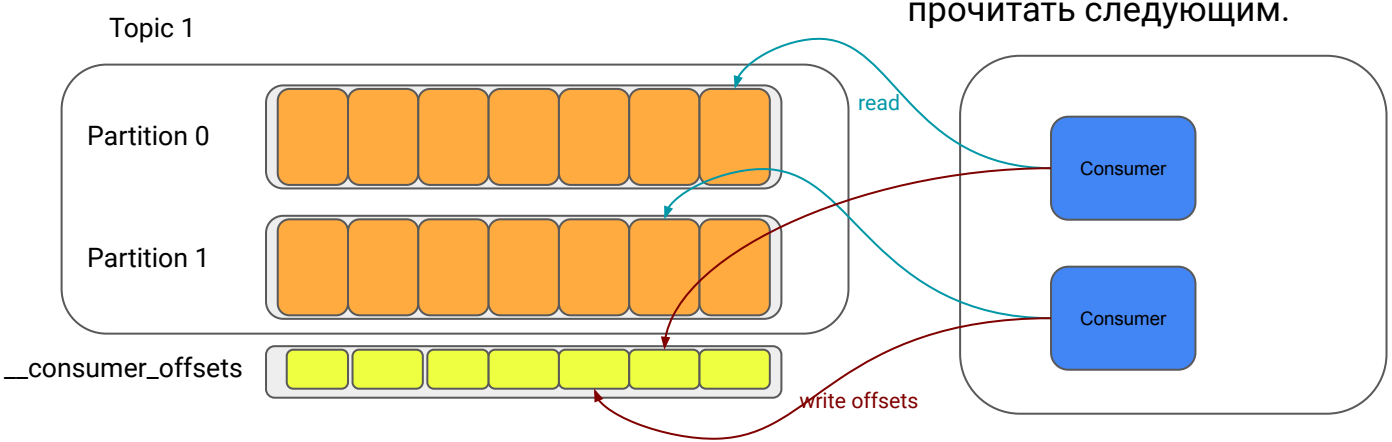
- Брокер отслеживает прогресс каждого Consumer;
- **Auto-commit** - оффсеты автоматически фиксируются в топике `__consumer_offsets;`



Offsets

Offset - это порядковый номер сообщения в партиции.

- Брокер отслеживает прогресс каждого Consumer;
- **Auto-commit** - оффсеты автоматически фиксируются в топике `__consumer_offsets;`
- В топик записывается оффсет, который нужно прочитать следующим.



Ручная фиксация оффсетов

- `enable.auto.commit=false`

Ручная фиксация оффсетов

- `enable.auto.commit=false`
- Два варианта фиксации:
  - `Consumer.commitSync()` - дожидается записи оффсетов в `__consumer_offsets`

Ручная фиксация оффсетов

- `enable.auto.commit=false`
- Два варианта фиксации:
  - `Consumer.commitSync()` - дожидается записи оффсетов в `__consumer_offsets`
  - `Consumer.commitAsync()` - не дожидается записи, сразу же выполняет код ниже. Можно добавить `callback`.

Ручная фиксация оффсетов

- `enable.auto.commit=false`
- Два варианта фиксации:
  - `Consumer.commitSync()` - дожидается записи оффсетов в `__consumer_offsets`
  - `Consumer.commitAsync()` - не дожидается записи, сразу же выполняет код ниже. Можно добавить `callback`.
- В методах можно указать конкретный топик, партицию, оффсет.

Ручная фиксация оффсетов

- `enable.auto.commit=false`
- Два варианта фиксации:
  - `Consumer.commitSync()` - дожидается записи оффсетов в `__consumer_offsets`;
  - `Consumer.commitAsync()` - не дожидается записи, сразу же выполняет код ниже. Можно добавить `callback`.
- В методах можно указать конкретный топик, партицию, оффсет.
- Можно совмещать оба метода:
  - `Consumer.commitAsync()` - для основного чтения;

Ручная фиксация оффсетов

- `enable.auto.commit=false`
- Два варианта фиксации:
  - `Consumer.commitSync()` - дожидается записи оффсетов в `__consumer_offsets`;
  - `Consumer.commitAsync()` - не дожидается записи, сразу же выполняет код ниже. Можно добавить `callback`.
- В методах можно указать конкретный топик, партицию, оффсет.
- Можно совмещать оба метода:
  - `Consumer.commitAsync()` - для основного чтения;
  - `Consumer.commitSync()` - перед закрытием `Consumer` (конструкция `finally`).

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Конфигурация

Основные настройки Consumer

- `bootstrap.servers` – адреса брокера кластера Kafka, например:
  `kafka-1:9092, kafka-2:9092, kafka-3:9092`

Основные настройки Consumer

- `bootstrap.servers` – адреса брокера кластера Kafka, например:
  `kafka-1:9092, kafka-2:9092, kafka-3:9092`
- `key.deserializer` - Класс для сериализации ключа сообщения. Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`

Основные настройки Consumer

- `bootstrap.servers` – адреса брокера кластера Kafka, например:
  `kafka-1:9092, kafka-2:9092, kafka-3:9092`
- `key.deserializer` - Класс для сериализации ключа сообщения. Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`
- `value.deserializer` – Класс для сериализации самого сообщения.Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`

Основные настройки Consumer

- `bootstrap.servers` – адреса брокера кластера Kafka, например:
  `kafka-1:9092, kafka-2:9092, kafka-3:9092`
- `key.deserializer` - Класс для сериализации ключа сообщения. Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`
- `value.deserializer` – Класс для сериализации самого сообщения.Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`
- `group.id` – уникальный идентификатор ConsumerGroup

Основные настройки Consumer

- `bootstrap.servers` – адреса брокера кластера Kafka, например: `kafka-1:9092, kafka-2:9092, kafka-3:9092`
- `key.deserializer` - Класс для сериализации ключа сообщения. Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`
- `value.deserializer` – Класс для сериализации самого сообщения.Должен применять интерфейс `org.apache.kafka.common.serialization.Deserializer`
- `group.id` – уникальный идентификатор ConsumerGroup
- `enable.auto.commit` – Автоматически фиксировать чтение оффсетов с периодичностью, указанной в `auto.commit.interval.ms` (по умолчанию 5000 ms)

Основные настройки Consumer.poll()

- `max.partition.fetch.bytes` – максимальное количество данных, полученное за один вызов `.poll()` из каждой партиции (по умолчанию 1MB)

Основные настройки Consumer.poll()

- `max.partition.fetch.bytes` – максимальное количество данных, полученное за один вызов `.poll()` из каждой партиции (по умолчанию 1MB)
- `max.poll.partitions` - максимальное количество сообщений, полученное за один вызов `.poll()` из всех партиций (по умолчанию 500).

1. Создание объекта Properties и KafkaConsumer

```
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.KafkaConsumer;

import java.util.Properties;

 Properties props = new Properties();

 props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "broker1:9092,broker2:9092");
 props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
         org.apache.kafka.common.serialization.StringDeserializer.class);
 props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
         org.apache.kafka.common.serialization.StringDeserializer.class);

 KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);
```

Gist с кодом

2. Подписка на топики

```
consumer.subscribe(Arrays.asList("topic1", "topic2"));
```

Можно подписаться на несколько топиков сразу

Gist с кодом

3. Чтение сообщений

Сколько времени ждать, если нет новых сообщений

```
while(true) {
    ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s",
                record.offset(), record.key(), record.value());
}
```

Gist с кодом

4. Закрытие Consumer

```
try {
    while (true) {
        ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
        for (ConsumerRecord<String, String> record : records)
            System.out.printf("offset = %d, key = %s, value = %s",
                    record.offset(), record.key(), record.value());
    }
} finally {
    consumer.close();
}
```

Gist с кодом

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Тезисы

Подведем итоги

1. Consumer API позволяет читать сообщения из Kafka
2. Возможно масштабировать чтение при помощи ConsumerGroup
3. Имеет много конфигураций - каждая по-своему влияет на производительность приложения

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Цели вебинара

После занятия вы сможете

1. Объяснить, что такое ConsumerGroups и зачем они нужны
2. Использовать Consumer API для отслеживания оффсетов
3. Настраивать Consumer

Рефлексия

С какими основными мыслями
и инсайтами уходите с вебинара?

Как будете применять на практике то,
что узнали на вебинаре?

Следующий вебинар

Admin API

- Ссылка на вебинар будет в ЛК за 15 минут
- Материалы к занятию в ЛК — можно изучать
- Обязательный материал обозначен красной лентой

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Чащина Александра

Big Data Engineer