OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Знакомство с Apache Kafka.

Основные компоненты, варианты развертывания, экосистема, сценарии использования

Непомнящий Евгений

Разработчик Java/Kotlin IT-Sense

@evgeniyN

Преподаватель

Непомнящий Евгений

- 15 лет программировал контроллеры на C++ и руководил отделом разработки
- 5 лет пишу на Java и Kotlin
- Последнее время работаю в IT-Sense

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

- Подготовка
- Сообщения и топики
- Партиции
- Репликация
- Гарантии
- API

Цели вебинара

После занятия вы сможете

1. Ознакомиться с основными концепциями Kafka
2. Запустить Kafka через docker-compose
3. Ознакомиться с утилитами для работы с Kafka
4. Отправить и получить сообщение вручную
5. Отправить и получить сообщение через API из программы на Java

Смысл

Зачем вам это уметь

1. Это второе вводное занятие. Перед тем, как перейти к более углубленным темам, надо понять основы
2. Возможно вы все это знаете. Тогда давайте просто “сверим часы”

Подготовка

Задание 1

Запустите kafka.

Репозитарий: https://github.com/OtusTeam/OTUS-Kafka

Мы будем использовать два приложения - kafka и zookeeper и запускать через docker-compose.
Это не обязательно, вы можете запустить kafka любым другим способом.

До занятия выполните команду
docker compose up -d (в папке lesson-02/kafka, где лежит docker-compose.yml)
первый запуск может занимать много времени.

Задание 2

Запустите приложение

Я буду использовать IntelliJ Idea для демонстрации. Вы можете использовать community edition (https://www.jetbrains.com/idea/download). Это не обязательно, можете использовать любую ide или обычный блокнот.

Вам нужна JDK 17. Например можно взять отсюда: https://bell-sw.com/pages/downloads/. Выбираете JDK17 LTS, свою ОС, standart edition.

Архив достаточно распаковать и настроить переменную окружения JAVA_HOME на папку, где находятся папки bin, conf и т.п. из архива.

git clone https://github.com/OtusTeam/OTUS-Kafka.git

Далее зайдите в папку lesson2 и запустите gradlew build - первый запуск может занимать много времени.

Kafka

Kafka – это система, реализующая распределенный реплицируемый лог сообщений

- Распределенный – лог (топик) не целиком хранится на одной машине, а разбит на несколько секций (партиций), которые лежат на разных машинах
- Реплицируемый -- логи хранятся в нескольких копиях, на случай отказов оборудования
- Лог – это упорядоченная последовательность сообщений

Сообщения и топики

Что такое сообщение

- Ключ
  - опционален
  - массив байт
- Заголовки
  - опциональны
  - набор пар “ключ: значение”
  - ключ - строка
  - значение - массив байт
- Тело
  - массив байт

Что в теле?

- JSON
- XML
- Protobuf
- Thrift
- Avro

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
struct ListBonks {
  1: list<Bonk> bonk
}
struct NestedListsBonk {
  1: list<list<list<Bonk>>> bonk
}

struct BoolTest {
  1: optional bool b = true;
  2: optional string s = "true";
}
```

```
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Топик

Топик – это некоторый аналог таблички в БД.

- Топик – это чащего всего набор событий одного типа (customerCreated, notificationSent и т.д)
- Если вам нужно получить гарантии порядка, относящиеся к одному бизнес процессу, то в топик можно записывать сообщения (события) разных типов: customerCreated, customerInvoicePaid и т.д

Слишком большое количество (больше 1000) топиков (и партиций) в одном брокере приводит к снижению производительности. Поэтому сильно дробить на топики не стоит.

Топик

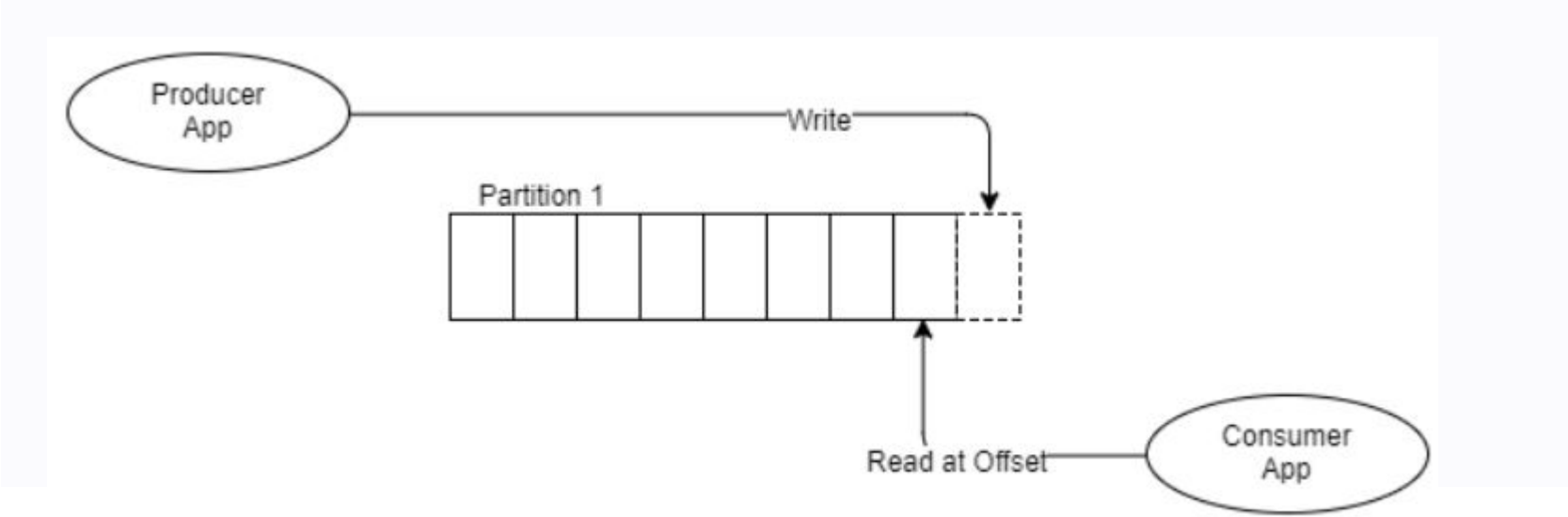
Топик – это набор неструктурированных сообщений (без схемы).

Для кафки – **сообщение** – это просто набор байт. Поверх которых можно использовать разные протоколы – JSON, Avro, Protobuf и т.д.

Чтобы появилась схема, используют внешние инструменты типа SchemaRegistry и StreamProcessing (KTable, KStream)

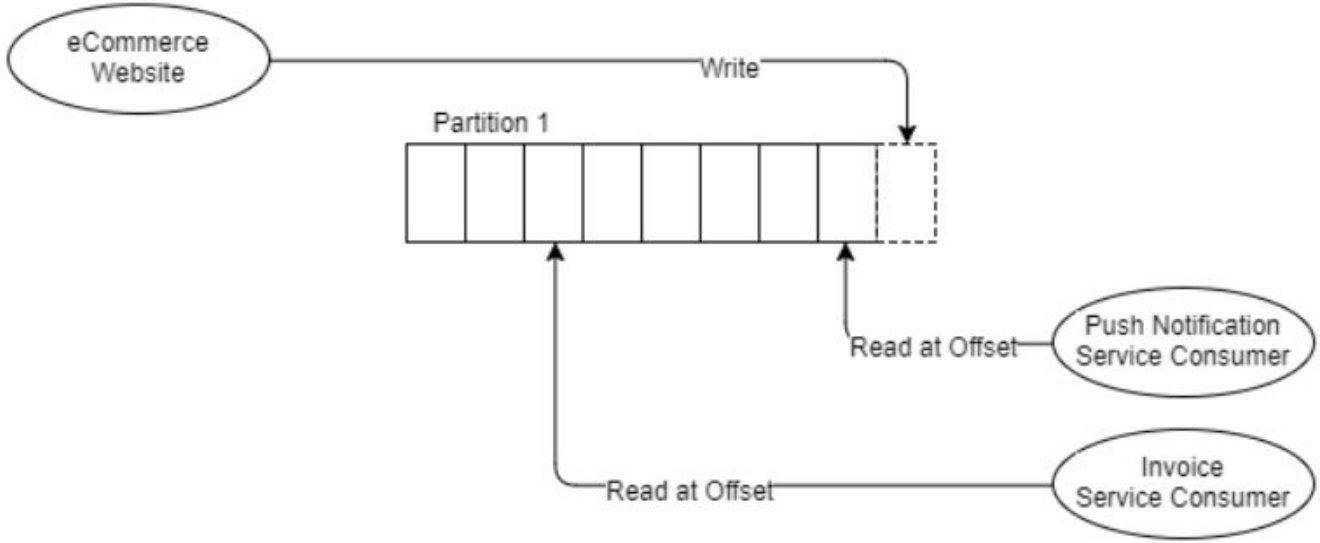
Топик

- Отправитель добавляет сообщение в распределенный лог сообщений
- Получатель (подписчик) читает из этого лога непрочтенные с прошлого раза записи (но может и любые другие)



Offset

Каждый получатель (consumer) хранит у себя offset – указатель на сообщение, которое он прочитал последним.



Демо

Создадим топик topic1 с настройками по умолчанию.

Отправим пару сообщений.

Посмотрим на них в kafdrop http://localhost:9000/

Задание 3

1. Создайте топик topic1 через kafdrop (http://localhost:9000/) или иным способом (например `kafka-topics`)
2. Отправьте пару сообщений через kafka-console-producer (см команду в actions.md)
3. Прочитайте сообщения с помощью kafdrop
4. Ставьте + в чат, когда сделаете (или если не можете / не хотите делать)

Задавайте вопросы, если есть проблемы

Вопросы?

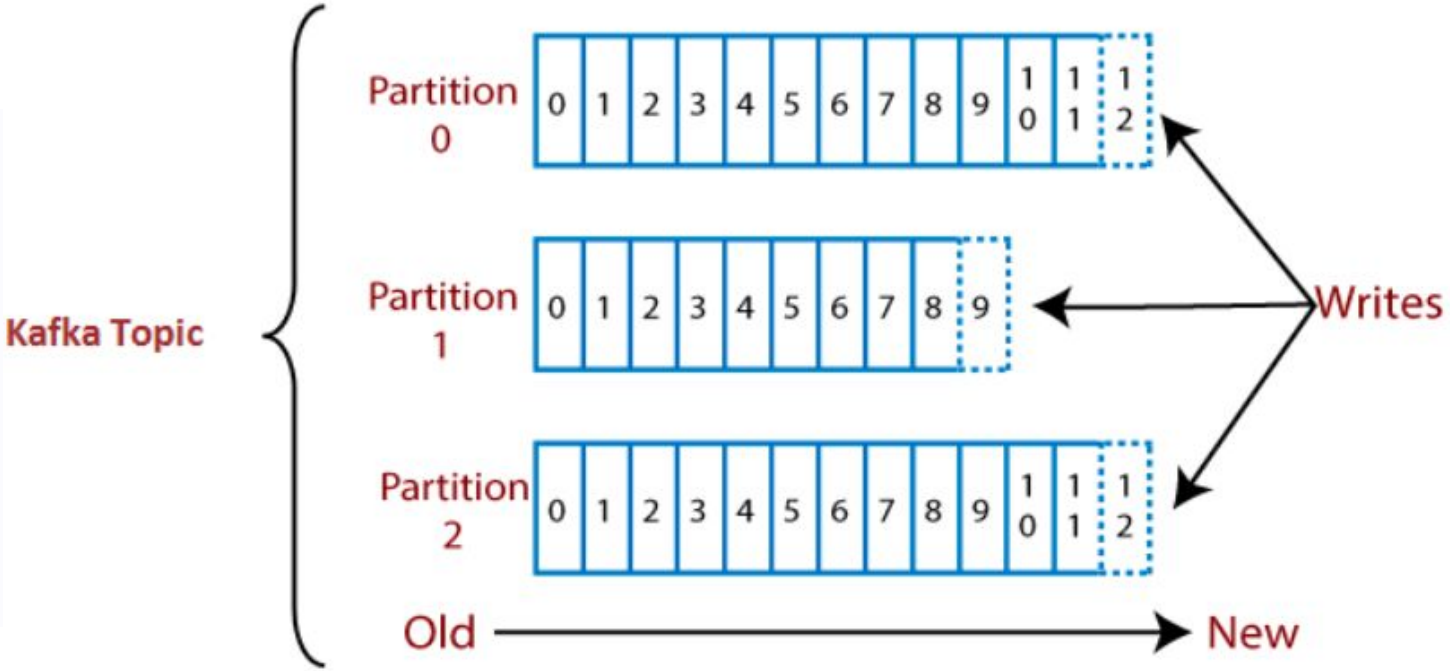
Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Партиции

Партиции

Топик разделен на несколько **партиций**.



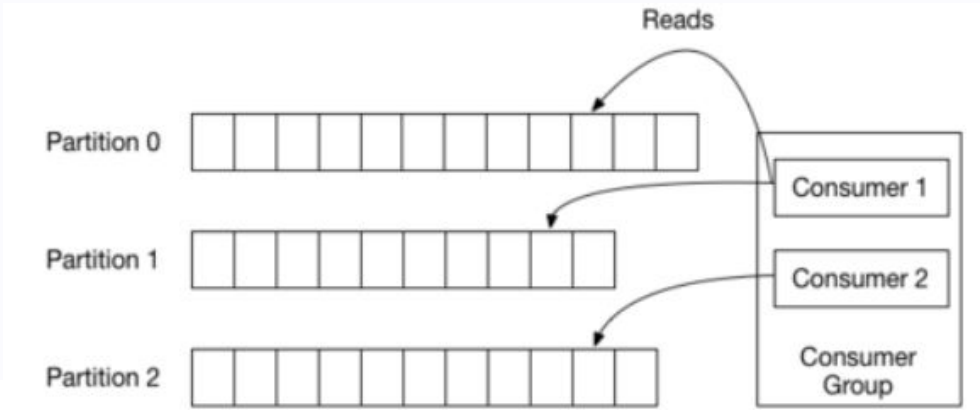
Зачем нужны партиции

- Партиции могут храниться на разных узлах, тем самым повышая скорость чтения и записи в топик (*распределенный лог*).
- Партиции завязаны на механизм доставки сообщения одному “логическому потребителю”

Consumer groups

Группа получателей или группа консьюмеров – это один логический консьюмер, который представляет из себя многопоточное или распределенное приложение. **И к каждой партиции привязан только один консьюмер.**

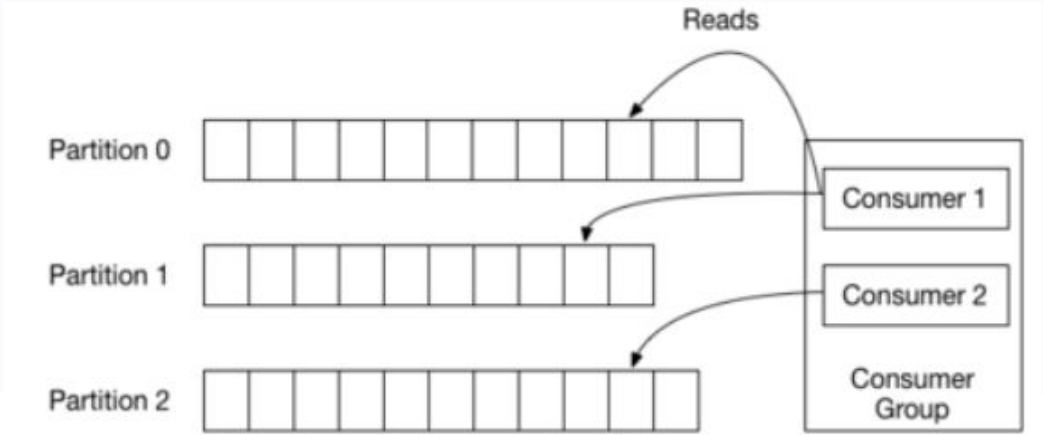
Почему? Потому что каждое сообщение должно быть доставлено этому логическому консьюмеру только один раз. Как это обеспечить, если консьюмер читает по смещению?



Consumer groups

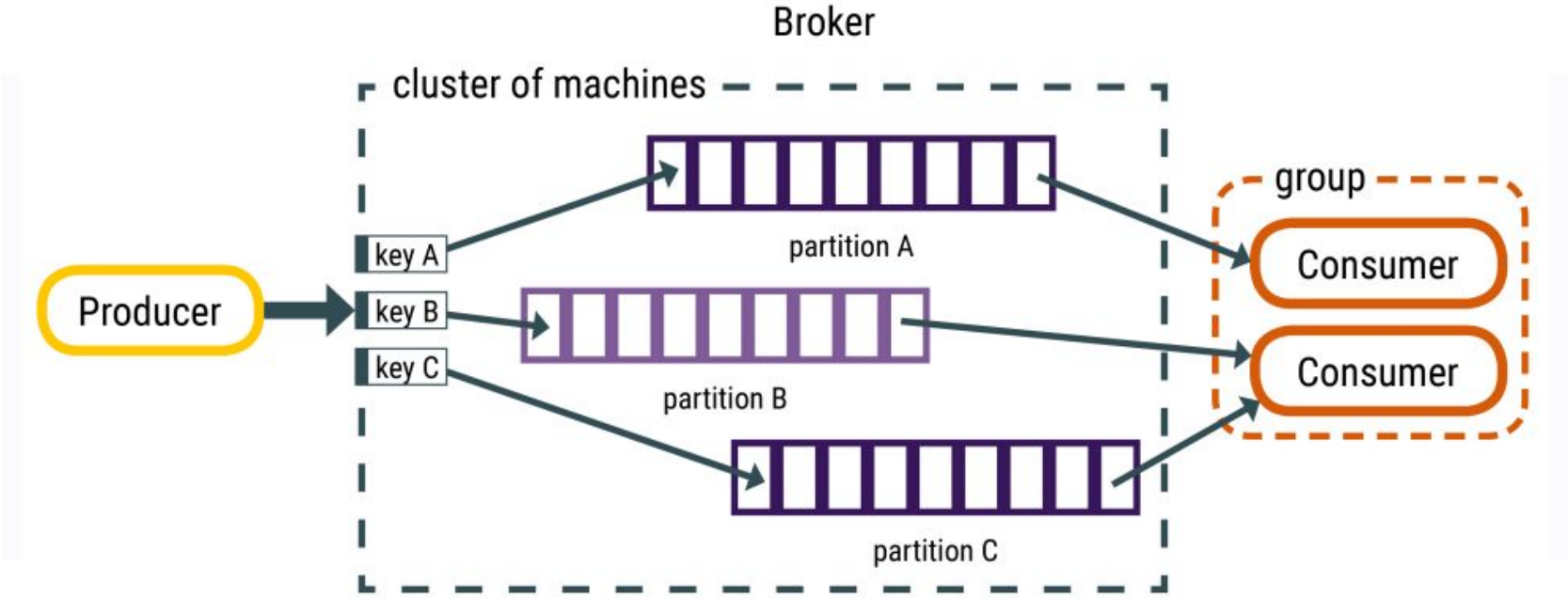
Для каждой consumer group, которая читает топик выделяется

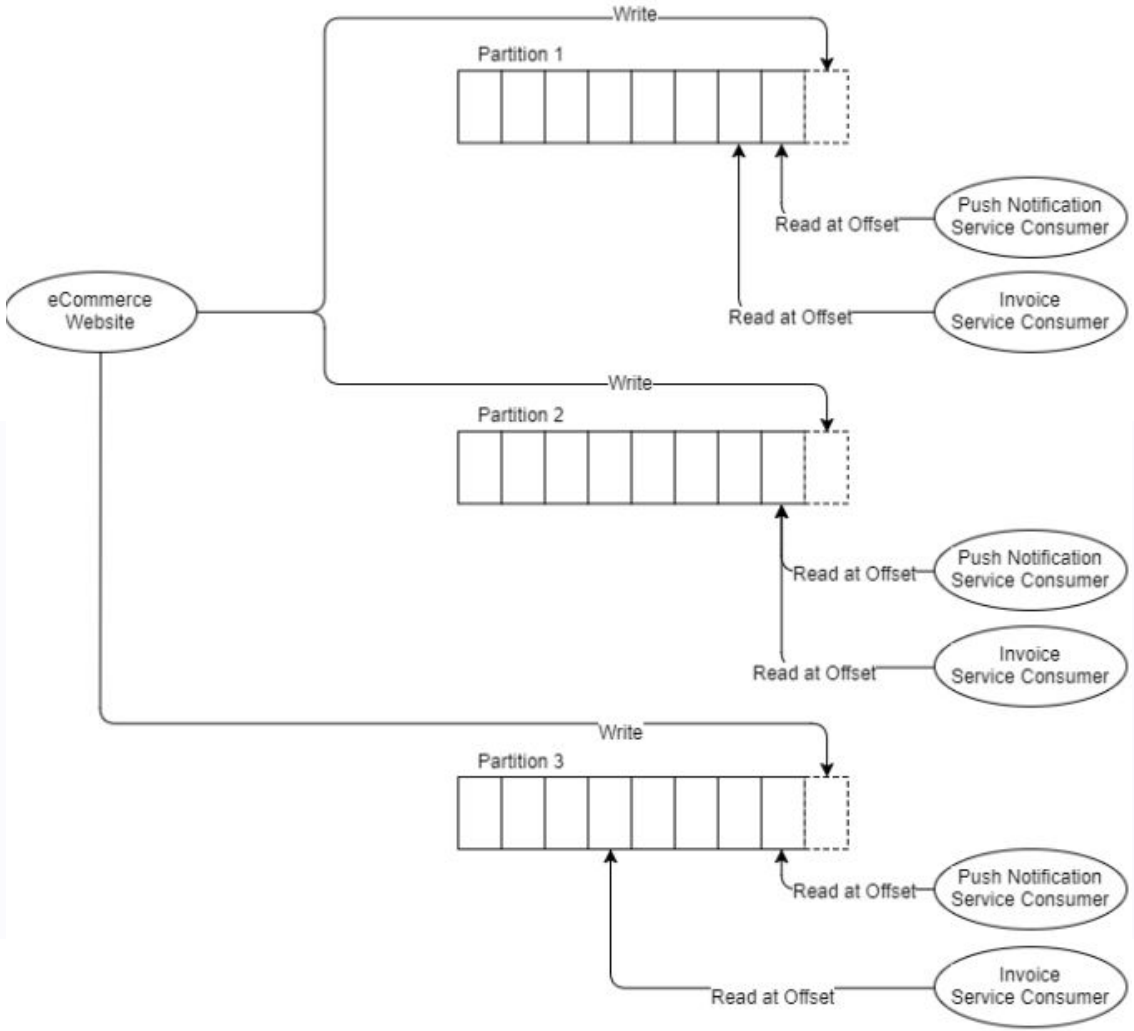
- **Group coordinator** – это один из брокеров кафки назначает себя координатором группы consumerов и отвечает за состав группы и живость членов группы.
- **Group leader** – это один из консьюмеров в группе, назначается случайным образом group coordinator-ом, и потом group leader распределяет consumer-ов по партициям



Ключ

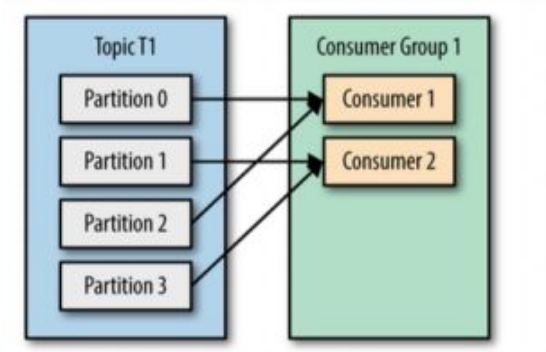
У сообщения в kafka может быть ключ - **key**. Если есть ключ, то тогда партиционирование происходит таким образом, чтобы сообщения с одним ключом попадали в одну партицию

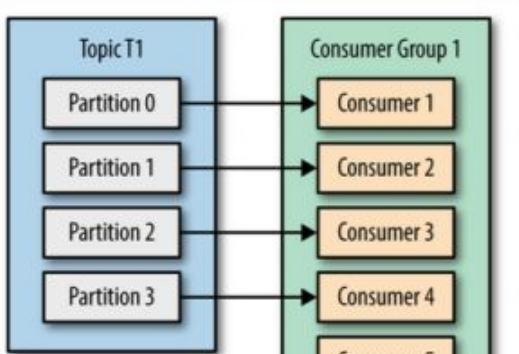


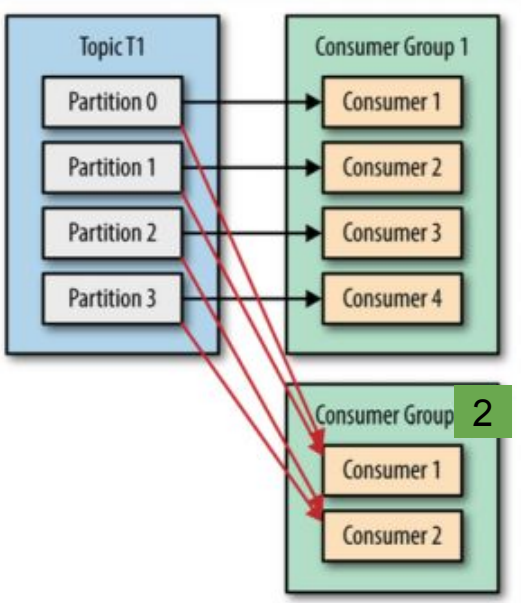

Write
Partition 1
Read at Offset
Push Notification Service Consumer
eCommerce Website
Read at Offset
Invoice Service Consumer
Write
Partition 2
Read at Offset
Push Notification Service Consumer
Read at Offset
Invoice Service Consumer
Write
Partition 3
Read at Offset
Push Notification Service Consumer
Read at Offset
Invoice Service Consumer

Consumer и partition

У каждой партиции должен быть только один конкурирующий консьюмер. Один консьюмер может читать из нескольких партиций.





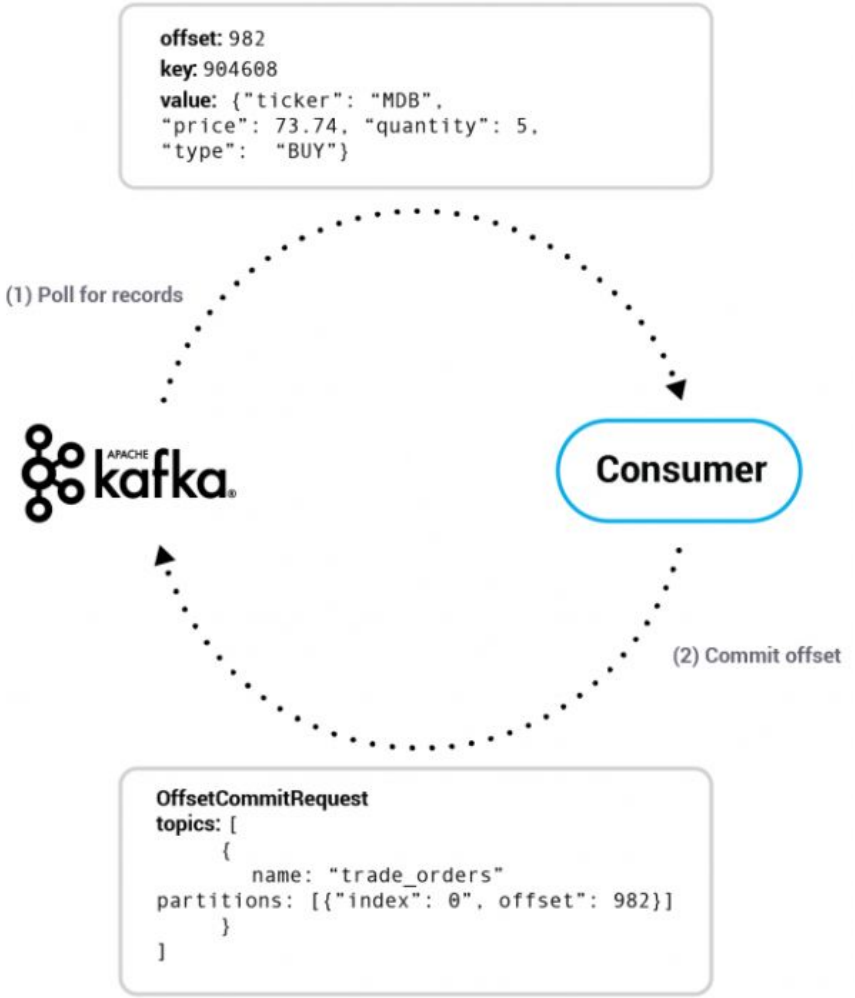


Offset и Commit offset

offset – метка сообщения, которое консьюмер прочитал.

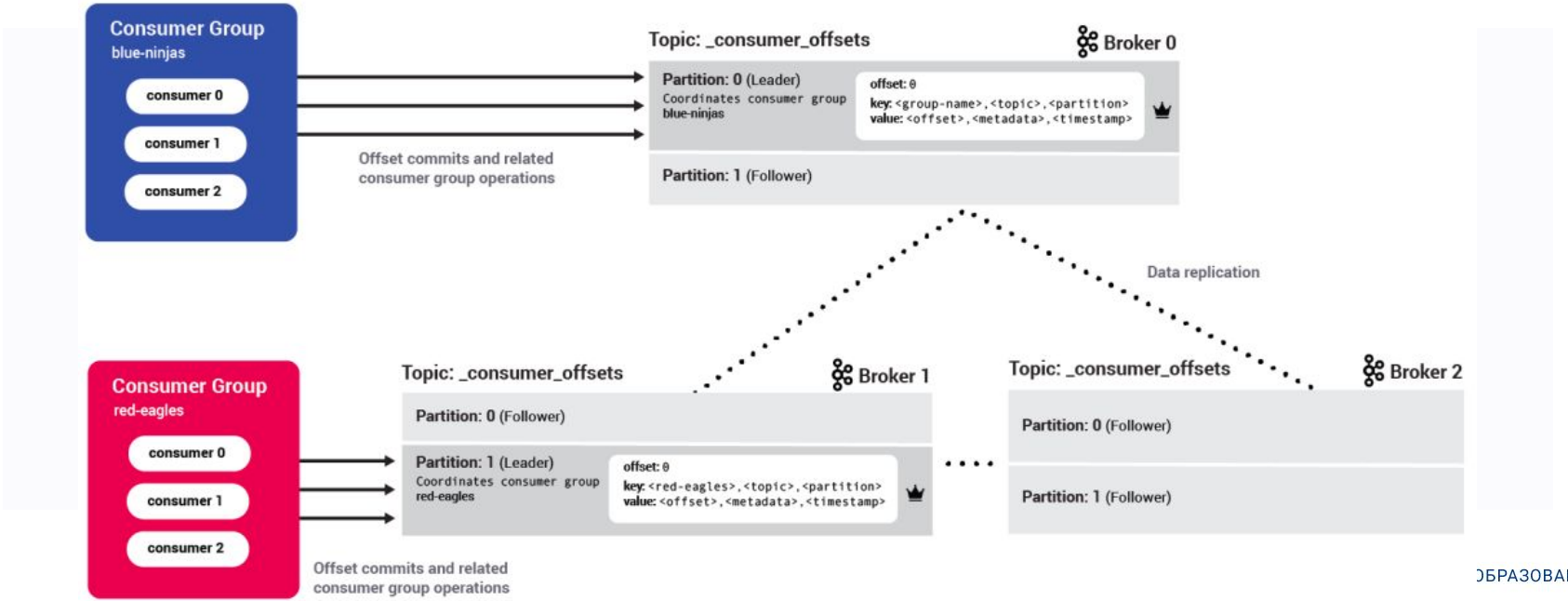
Для того, чтобы понимать, какие сообщения consumer (получатель) обработал, используются

commit offset – последняя обработанная запись



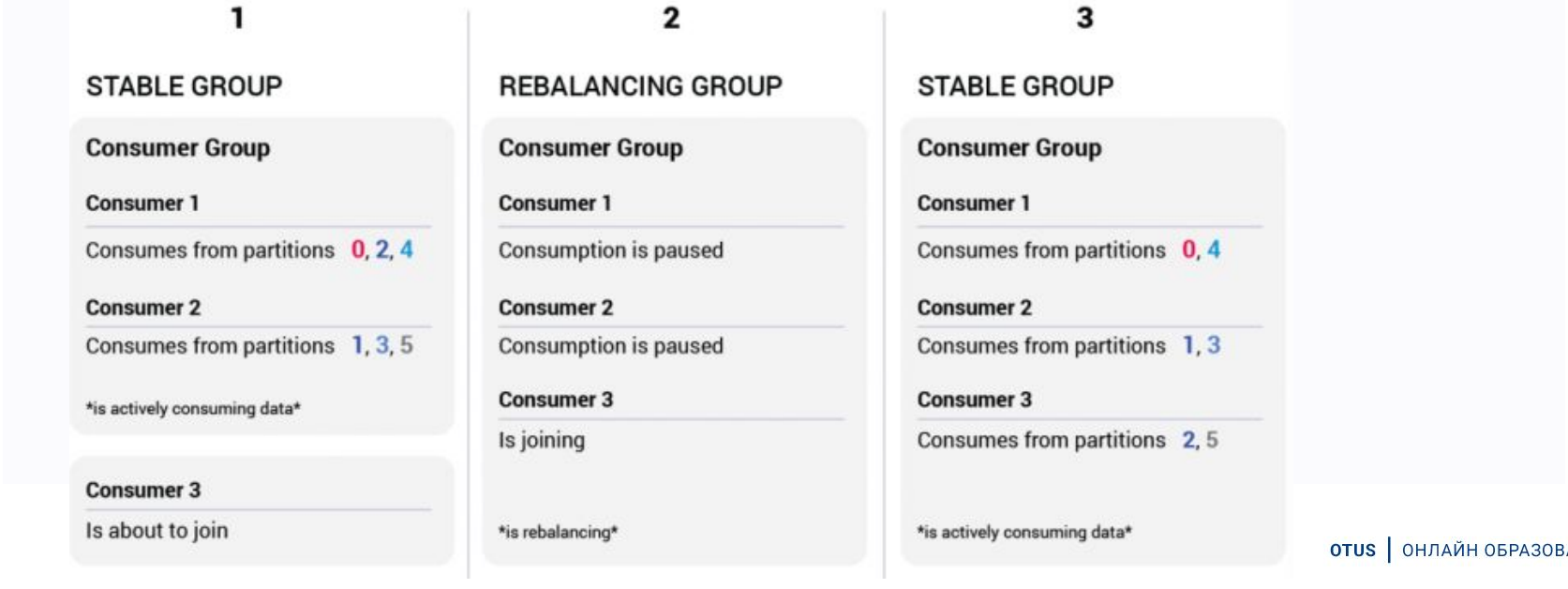
Commit offset

Commited offsets хранятся в топике __consumer_offsets, который находится у координатора группы (и реплицируется на остальные брокеры). При этом для разных consumer group и одного топика координатор может быть разным.



Перебалансировка

Иногда хочется добавить или удалить нового consumer-а в группу, или один из консьюмеров падает. В этом случае, consumer group переходит в состояние перебалансировки, во время которого ни один из консьюмеров не читает сообщения из топиков.



Демо

Запустим два получателя из topic1 с разными группами.

Запустим отправителя с парсингом ключа и отправим пару сообщений с разными ключами.

Почему так?

Теперь тоже самое, но получатели из одной группы.

Почему так?

Далее создадим topic2 с 2 партициями и повторим эксперимент

Почему так?

Отправим пару сообщений с одним ключом.

Почему так?

Задание 4

1. Создайте топик topic2 через kafdrop (http://localhost:9000/) с двумя партициями
2. Запустите два консьюмера с одной группой и консьюмер с другой группой
3. Отправьте пару сообщений с разными ключами, посмотрите, кто их получит
4. Ставьте + в чат, когда сделаете (или если не можете / не хотите делать)

Задавайте вопросы, если есть проблемы

Вопросы?

Ставим “+”,
если вопросы есть

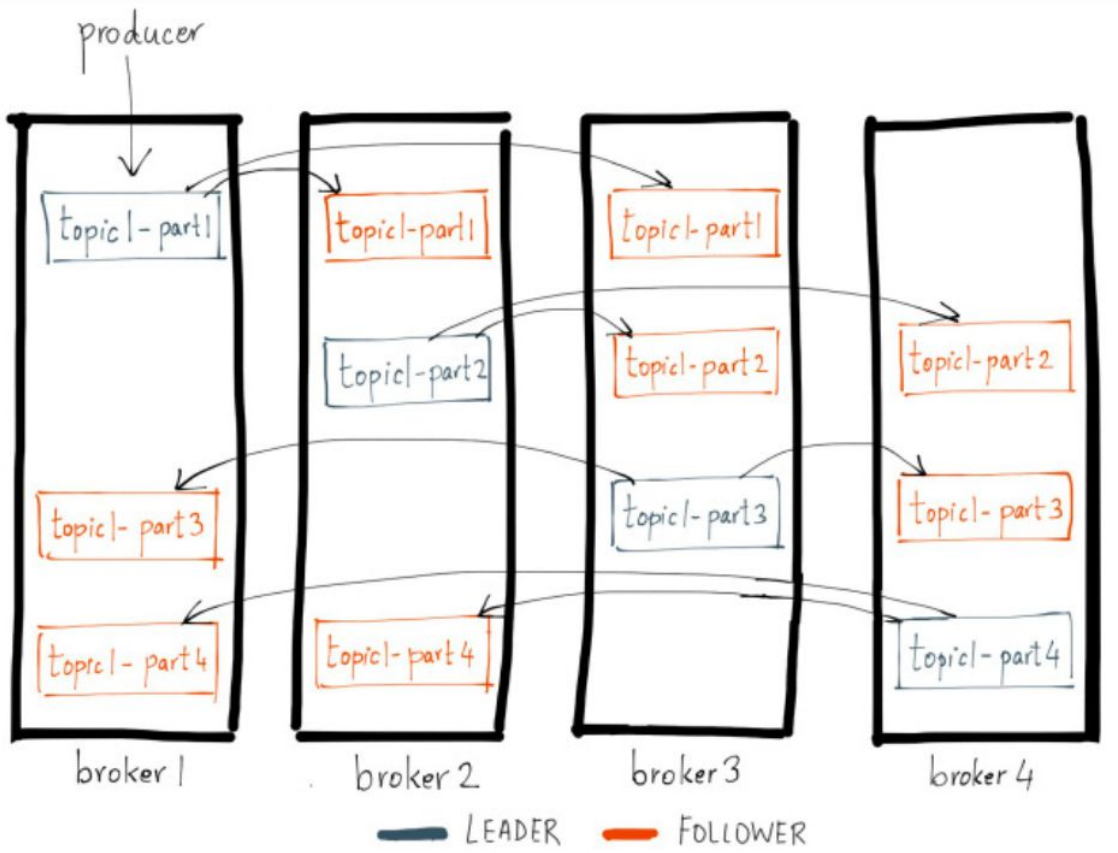
Ставим “–”,
если вопросов нет

Репликация

Репликация

В Kafka поддерживается репликация. Каждая партиция в топике может иметь несколько реплик.

В рамках каждого брокера (инстанса Кафки) может быть до тысячи реплик, относящихся к разных топикам.

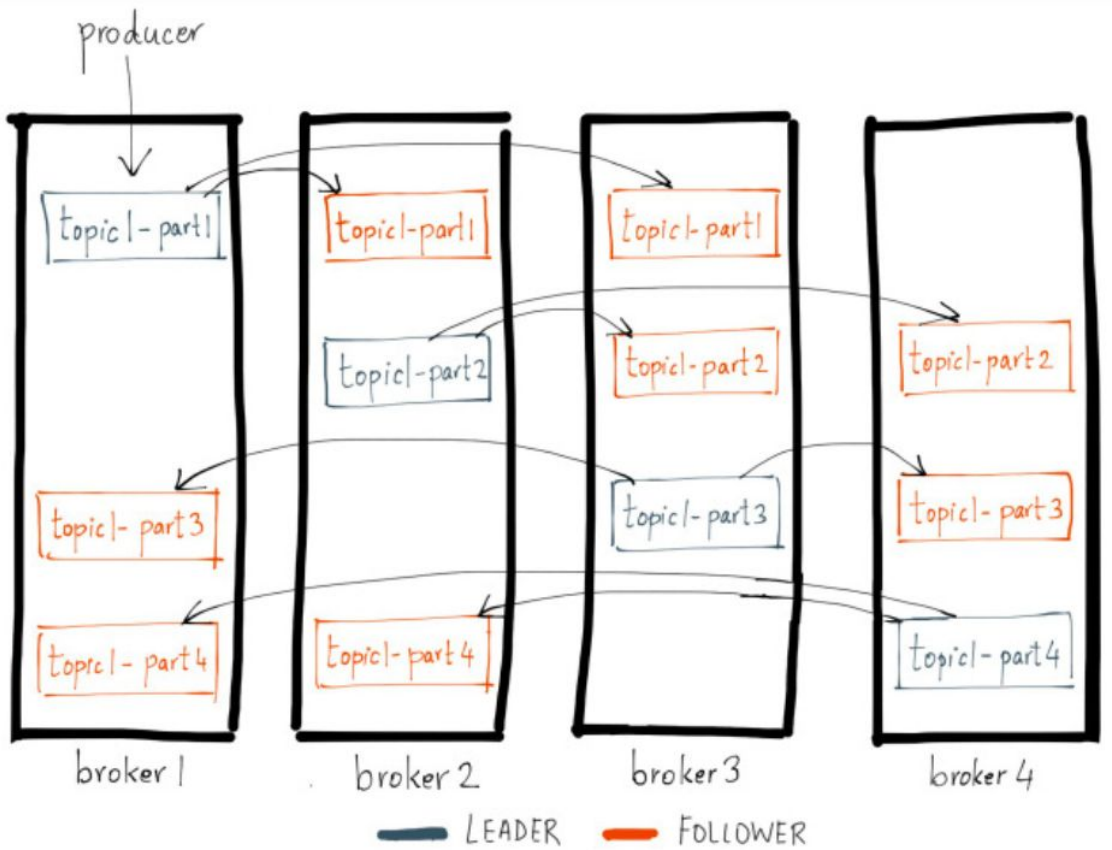


Репликация

Авторы Kafka советуют

- фактор репликации 1, если вы ок, с тем, что можете эти данные потерять,
- фактор репликации 3, если хотите, чтобы данные остались в сохранности.

Фактор репликации 2 не советуют использовать из-за того, что в этом случае могут быть проблемы с тем, что кластер Kafka может развалится в некоторых ситуациях на старых версиях



Гарантии

Гарантии в Kafka

- Если producer отправил сообщение B после сообщения A в тот же самый топик, в той же самой партиции, то offset B будет больше A.
- Если producer отправил сообщение и все in sync replica-и подтвердили, что они сохранили у себя, то тогда сообщение можно считать закоммиченными.
- Закоммиченные сообщения не будут потеряны, пока хотя бы одна реплика жива
- Consumer-ы могут читать только закоммиченные сообщения.

Надежная отправка

При отправка сообщения producer может его отправить с параметром acks.

- acks = 0 означает, что если произошла отправка по сети, то отправка считается успешной (например, сообщение может быть потеряно во время выборов)
- acks = 1 означает, что отправка считается успешной, если лидер записал это сообщение на диск
- acks = all означает, что отправка считается успешной, если лидер и все in-sync-replica-и подтвердили приемку сообщения

Надежная отправка

Пусть будет topic с фактором репликации 3. В случае, если producer отправляет сообщения с параметром acks=1, то это может привести к потери сообщения.

Например, если лидер получил это сообщение, ответил producer-у, что все ок, но в этот момент упал, не успев отправить сообщение репликам. Тогда из оставшихся реплик будет выбран лидер, в котором этого сообщения не будет.

Надежная отправка

В случае ошибки отправки , клиент (producer) со своей стороны должен повторить запрос через некоторое время. И это может приводить к дублированию сообщений.

Для того, чтобы такого дублирования не было, Kafka умеет в идемпотентные сообщения, и рекламирует это как exactly once delivery гарантии.

Надежное получение

Есть разница между **commited** сообщениями и **__commited_offset**

- Commited – сообщения, которые были записаны на всех in sync replicax
- Commited offset – это сообщения, которые были обработаны в рамках consumer group на каждой из партиций

Надежное получение

В случае если стоит опция auto_commit=true, то тогда консьюмеры коммитят последний полученный из poll offset раз в 5 секунд (интервал настраивается). Это означает, что если консьюмер упадет между двумя вызовами коммита, он успеет обработать какие-то сообщения, и при перебалансировке они снова будут обработаны.

Чтобы такого не случилось, можно синхронно коммитить после того, как сообщения были обработаны.

Pull - модель

Особенности push модели в реализации Kafka:

- Чуть хуже latency за счет ожидания перед каждым запросом данных
- Лучше сразу думать про масштабирование: добавление партиций приводит к ребалансингу
- Гарантированный порядок сообщений в рамках партиции

Kafka vs RabbitMq

- Kafka
  - Высокая пропускная способность (позволяет отправлять большие объемы данных достаточно быстро)
  - Хранит сообщения даже после чтения - можно перечитать
  - Репликация из коробки
  - Сложнее в настройке и запуске
- RabbitMq
  - Лучше Latency (быстрее проходят отдельные сообщения)
  - Значительно гибче роутинг (exchange / bind / queue)

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

API

Java

Есть те, кто не знает Java / gradle?

Kafka API

- Producer API - kafka-clients
- Consumer API - kafka-clients
- Streams API - kafka-streams
- Admin API - kafka-clients
- Connect API

Producer API

- KafkaProducer<Key, Value>
- Future<RecordMetadata> send(ProducerRecord<K, V> record)
- Future<RecordMetadata> send(ProducerRecord<K, V> record, Callback callback)
- Отправка асинхронна
- Можно отправить принудительно и синхронно - void flush()
- Очень много параметров - см ProducerConfig

Consumer API

- KafkaConsumer<Key, Value>
- void subscribe(Collection<String> topics)
- ConsumerRecords<K, V> poll(final Duration timeout)
- Получение синхронно
- Может приехать 0, 1 или много сообщений
- Очень много параметров - см ConsumerConfig

Демо

- Ex1Producer, смотрим в kafdrop - базовая отправка
- Ex2Producer - отправка асинхронна + key=Integer
- Ex3Producer - принудительная отправка

Демо

- Ex4Consumer
  - group-1 topic1 topic2
  - group-1 topic1 topic2
  - group-2 topic1 topic2
- Ex4Producer
  - Отправим 1..10 в topic1
  - Отправим 1..10 в topic2
  - Отправим 11..12 (10 штук) в topic2
  - Грохнем второй консамер и сразу отправим 20..30 в topic2 - где??
  - Фух
  - Запустим group-1 topic1 topic2

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Рефлексия

Ключевые тезисы

1. Kafka - распределенный децентрализованный лог
2. Топик
3. Партиции - для распараллеливания чтения и ускорения записи
4. Offset
5. Consumer-API, Producer-API

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Непомнящий Евгений

Разработчик Java/ Kotlin IT-Sense

@evgeniyN