OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Kafka Producer API

Чащина Александра

Big Data Engineer

Преподаватель

Александра Чащина

Выпускница магистерской программы “Big Data” университета Париж-Сакле

- *Прошла сертификации Databricks Spark&Scala и DevOps Foundation*
- *5 лет опыта работы в сфере данных*
- *преподаватель курса Инженер Данных и Spark Developer в OTUS*

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

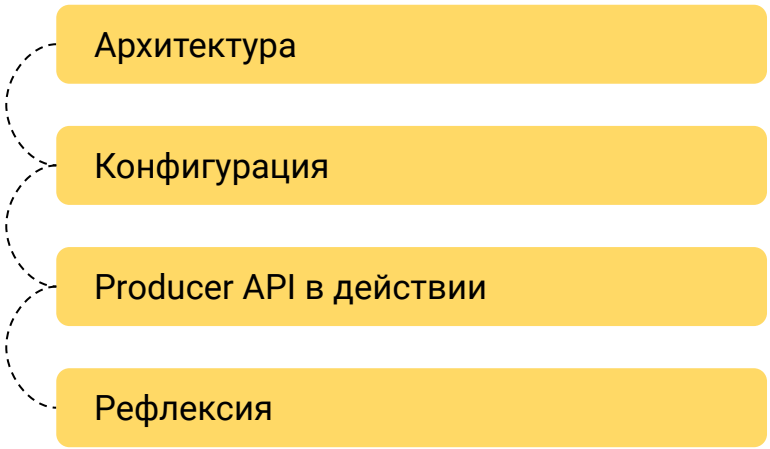
Документ

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. Понимать архитектуру Producer Kafka
2. Понимать механику партицирования топиков по ключу
3. Понимать разницу между различными типами подтверждений

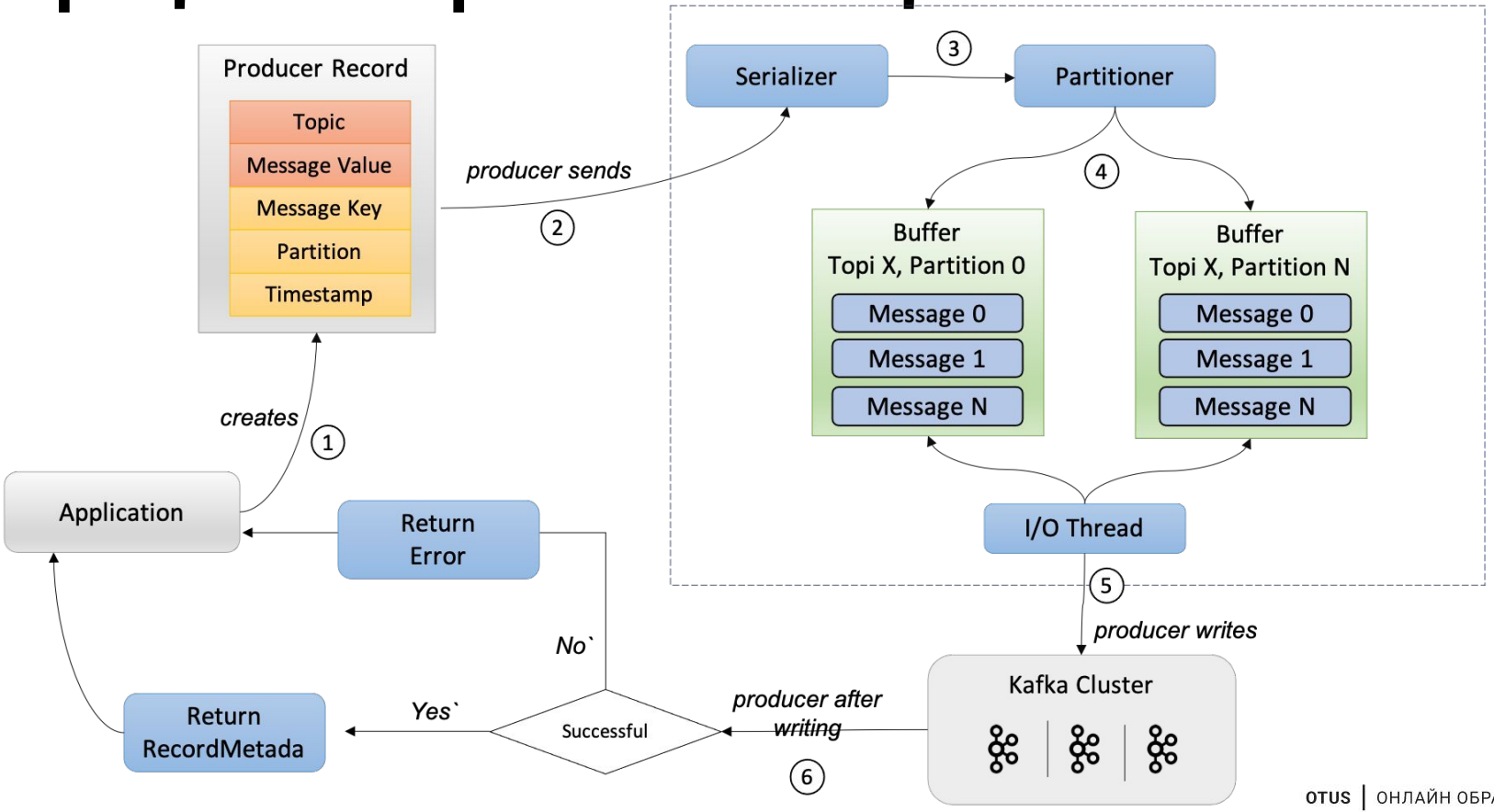
Смысл

Зачем вам это уметь

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

Архитектура Producer

Процесс отправки сообщений в Kafka



Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Обязательные поля:

1. **Topic** - название топика

Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Обязательные поля:

1. **Topic** - название топика
2. **Value** - непосредственно содержимое сообщения (в формате Avro, Json, Protobuf)

Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Обязательные поля:

1. **Topic** - название топика
2. **Value** - непосредственно содержимое сообщения (в формате Avro, Json, Protobuf)

Необязательные поля:

1. **Message Key** - ключ сообщения

Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Обязательные поля:

1. **Topic** - название топика
2. **Value** - непосредственно содержимое сообщения (в формате Avro, Json, Protobuf)

Необязательные поля:

1. **Message Key** - ключ сообщения
2. **Partition** - номер партиции

Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Обязательные поля:

1. **Topic** - название топика
2. **Value** - непосредственно содержимое сообщения (в формате Avro, Json, Protobuf)

Необязательные поля:

1. **Message Key** - ключ сообщения
2. **Partition** - номер партиции
3. **Timestamp** - отметка времени отправки сообщения

Producer Record

Producer Record - это данные, которые Producer отправляет в брокер Kafka.

Обязательные поля:

1. **Topic** - название топика
2. **Value** - непосредственно содержимое сообщения (в формате Avro, Json, Protobuf)

Необязательные поля:

1. **Message Key** - ключ сообщения
2. **Partition** - номер партиции
3. **Timestamp** - отметка времени отправки сообщения
4. **Headers** - заголовки сообщения в формате ключ-значение

Алгоритм выбора партиции

Указан верный номер партиции?	Указан ключ?	Партиция
Да	-	Партиция, номер которой указан в поле **Partition**
Нет	Да	Номер партиции = `hash(key) mod nb_partitions`
Нет	Нет	Определяется по алгоритму round-robin

Timestamp - отметка времени

Если не задан timestamp, то Producer добавит его автоматически, в зависимости от конфигурации топика:

Timestamp - отметка времени

Если не задан timestamp, то Producer добавит его автоматически, в зависимости от конфигурации топика:

1. **CreateTime** - timestamp соответствует времени отправки сообщения в Producer

Timestamp - отметка времени

Если не задан timestamp, то Producer добавит его автоматически, в зависимости от конфигурации топика:

1. **CreateTime** - timestamp соответствует времени отправки сообщения в Producer

2. **LogAppendTime** - timestamp соответствует времени доставки сообщения в брокер Kafka

Producer - дополнительные свойства

- Перед отправкой в брокер сообщения собираются в батчи для более оптимального использования сети

Producer - дополнительные свойства

- Перед отправкой в брокер сообщения собираются в батчи для более оптимального использования сети
- Один батч содержит сообщения только одной партиции

Producer - дополнительные свойства

- Перед отправкой в брокер сообщения собираются в батчи для более оптимального использования сети
- Один батч содержит сообщения только одной партиции
- Данные перед отправкой можно сжимать

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Конфигурация

Основные настройки Producer

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

Основные настройки Producer

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

Основные настройки Producer

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

Основные настройки Producer

- `bootstrap.servers` – адреса брокера кластера Kafka, например: `kafka-1:9092, kafka-2:9092, kafka-3:9092`
- `key.serializer` - Класс для сериализации ключа сообщения. Должен применять интерфейс `org.apache.kafka.common.serialization.Serializer`
- `value.serializer` – Класс для сериализации самого сообщения.Должен применять интерфейс `org.apache.kafka.common.serialization.Serializer`
- `compression.type` – Метод компрессации сообщений: *`none`*, `snappy`, `gzip`, `lz4`, `zstd`

Основные настройки Producer

- `bootstrap.servers` – адреса брокера кластера Kafka, например: `kafka-1:9092, kafka-2:9092, kafka-3:9092`
- `key.serializer` - Класс для сериализации ключа сообщения. Должен применять интерфейс `org.apache.kafka.common.serialization.Serializer`
- `value.serializer` – Класс для сериализации самого сообщения.Должен применять интерфейс `org.apache.kafka.common.serialization.Serializer`
- `compression.type` – Метод компрессации сообщений: *`none`*, `snappy`, `gzip`, `lz4`, `zstd`
- `acks` – Метод подтверждения доставки сообщений: `0`, *`1`*, `all (-1)`

Основные настройки Producer

- `delivery.timeout.ms` – Максимальное время после возвращения результата от метода `.send()` и перед тем, чтобы окончательно определить, отправлено ли сообщение или нет. По умолчанию `120000`

Основные настройки Producer

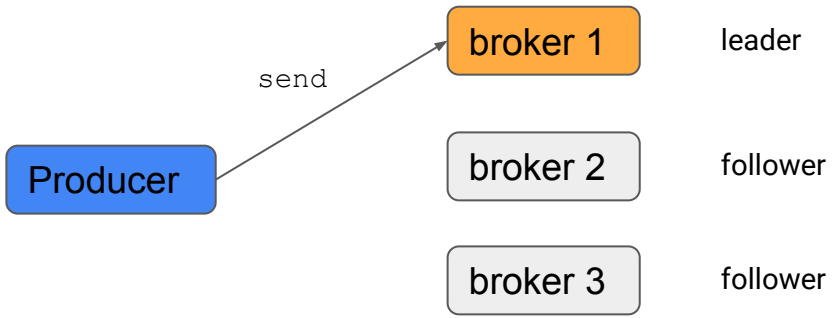
- `delivery.timeout.ms` – Максимальное время после возвращения результата от метода `.send()` и перед тем, чтобы окончательно определить, отправлено ли сообщение или нет. По умолчанию `120000`
- `batch.size` - Максимальный размер батча сообщений (в байтах). По умолчанию `16384`

Основные настройки Producer

- `delivery.timeout.ms` – Максимальное время после возвращения результата от метода `.send()` и перед тем, чтобы окончательно определить, отправлено ли сообщение или нет. По умолчанию `120000`
- `batch.size` - Максимальный размер батча сообщений (в байтах). По умолчанию `16384`
- `linger.ms` – Время, которое батч будет ждать новых сообщений перед отправкой. По умолчанию `0`

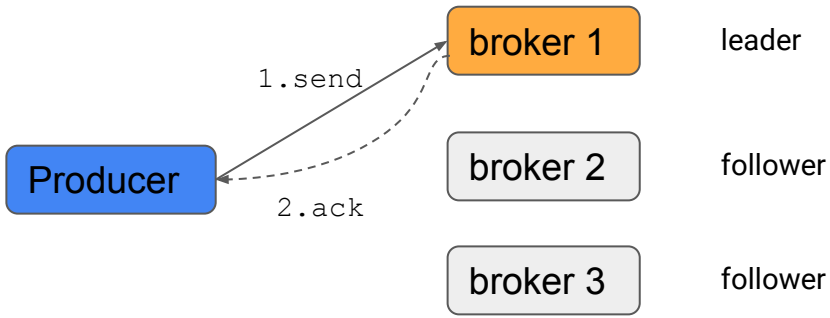
Acks - подтверждение доставки сообщения

`Acks = 0:` без подтверждения



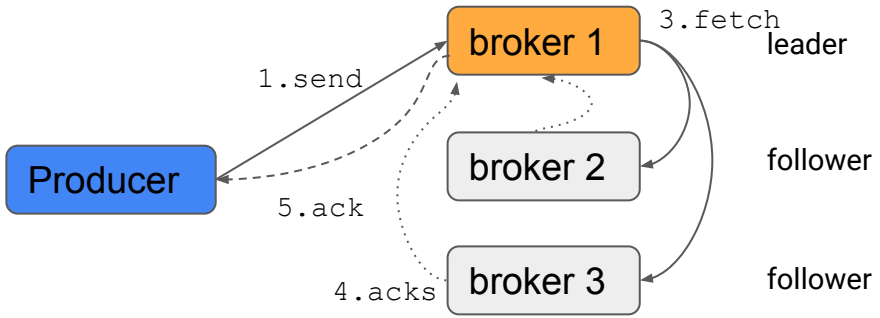
Acks - подтверждение доставки сообщения

`Acks = 1:` достаточно подтверждения от брокера-лидера

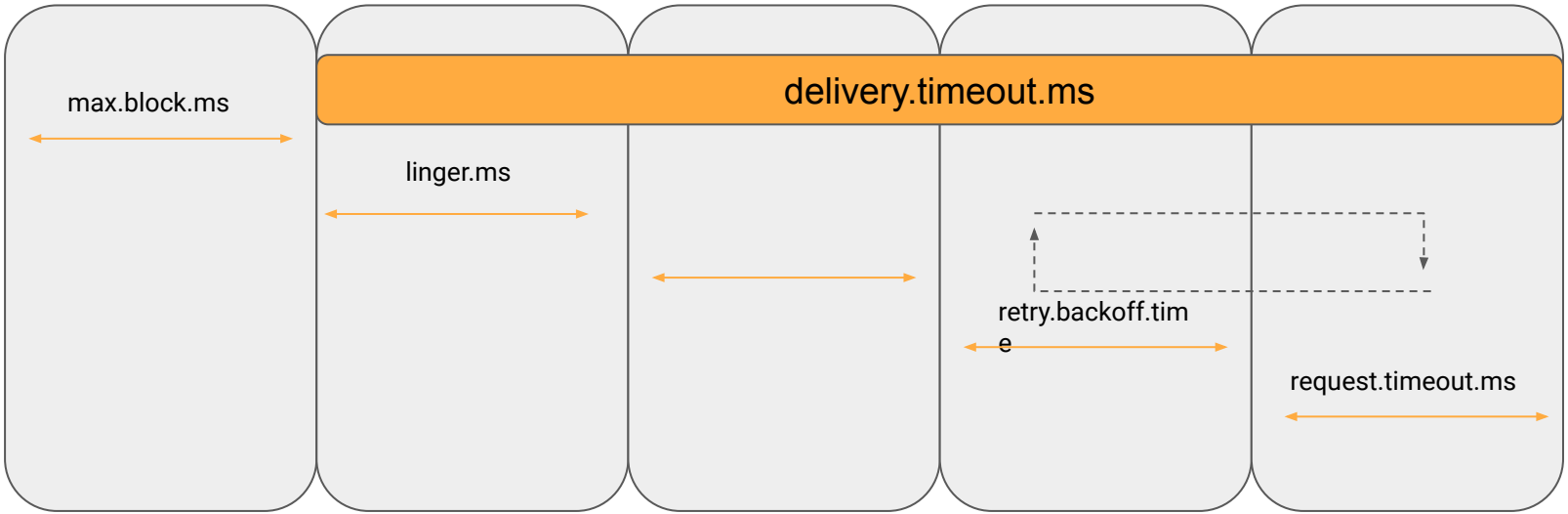


Acks - подтверждение доставки сообщения

`Acks = all:` необходимо подтверждение от всех брокеров



Delivery timeout и количество retries



Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Producer API

Kafka Producer API

Официальные:

- C/C++
- Go
- Java
- .NET
- Python
- Scala

Комьюнити:

- Erlang
- Groovy
- Haskell
- Kotlin
- Lua
- Node.js
- OCaml
- PHP
- Ruby
- Rust
- Tcl
- Swift

1. Создание объекта Properties и KafkaProducer

```
import org.apache.kafka.clients.producer.KafkaProducer;
import java.util.Properties;

Properties props = new Properties();

props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "broker1:9092,broker2:9092");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

Есть и другие форматы:
`ByteArraySerializer`,
`IntegerSerializer`,
`LongSerializer`,
`KafkaAvroSerializer`

Gist с кодом

1. Создание объекта Properties и KafkaProducer

```
import org.apache.kafka.clients.producer.KafkaProducer;
import java.util.Properties;

Properties props = new Properties();

props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "broker1:9092,broker2:9092");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

Properties можно задавать через файл `producer.properties` в следующем формате::

```
bootstrap.servers=broker1:9092, broker2:9092
```

Gist с кодом

1. Создание объекта Properties и KafkaProducer

```
import org.apache.kafka.clients.producer.KafkaProducer;
import java.util.Properties;

Properties props = new Properties();

props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "broker1:9092,broker2:9092");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

Хорошо использовать класс с константами - названиями конфигов: `org.apache.kafka.clients.producer.ProducerConfig.BOOTSTRAP_SERVERS_CONFIG`

Gist с кодом

2. Отправка сообщения

```
String k = "mykey";
String v = "myvalue";
ProducerRecord<String, String> record =
        new ProducerRecord<String, String>( topic: "myTopic", k, v);
producer.send(record);
```

`.send()` - это асинхронный метод: он не ждет ответа от брокера, сразу приступает к выполнению кода ниже

Gist с кодом

3. Обработка асинхронного ответа от брокера в Callback

```java
import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.RecordMetadata

public class CustomCallback implements Callback {

    public void onCompletion(RecordMetadata recordMetadata, Exception e) {
        if (e != null) {
            e.printStackTrace();
        } else {
            System.out.println("Message sent: " + recordMetadata.offset());
        }
    }
}
```

- `recordMeatadata = null`, если возникла ошибка
- `exception = null`, если ошибок нет

Gist с кодом

3. Обработка асинхронного ответа от брокера в Callback

```
String k = "mykey";
String v = "myvalue";
ProducerRecord<String, String> record =
        new ProducerRecord<String, String>( topic: "myTopic", k, v);
producer.send(record, new CustomCallback());
```

Gist с кодом

Блокирующий Producer

- `producer.send(record).get();` – дождаться отправки и доставки сообщения (synchronous producer)

Gist с кодом

Блокирующий Producer

- `producer.send(record).get();` – дождаться отправки и доставки сообщения (synchronous producer)
- `producer.flush();` – отправить все сообщения в буфере (каждую партицию по очереди)

Gist с кодом

4. Закрытие Producer

Дождаться отправки и доставки всех сообщений в буфере

```
producer.close();
```

```
producer.close(Duration.ofSeconds(60));
```

Закрыть Producer после 60 секунд, даже если какие-то сообщения не будут доставлены

Gist с кодом

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Тезисы

Подведем итоги

1. Producer API позволяет отправлять сообщения в Kafka
2. Доступен на многих языках
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. Понимать архитектуру Producer Kafka
2. Понимать механику партицирования топиков по ключу
3. Понимать разницу между различными типами подтверждений

Рефлексия

С какими основными мыслями
и инсайтами уходите с вебинара?

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

Следующий вебинар

Consumer 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