СБЕР

Школа Java Middle Developer

Kafka

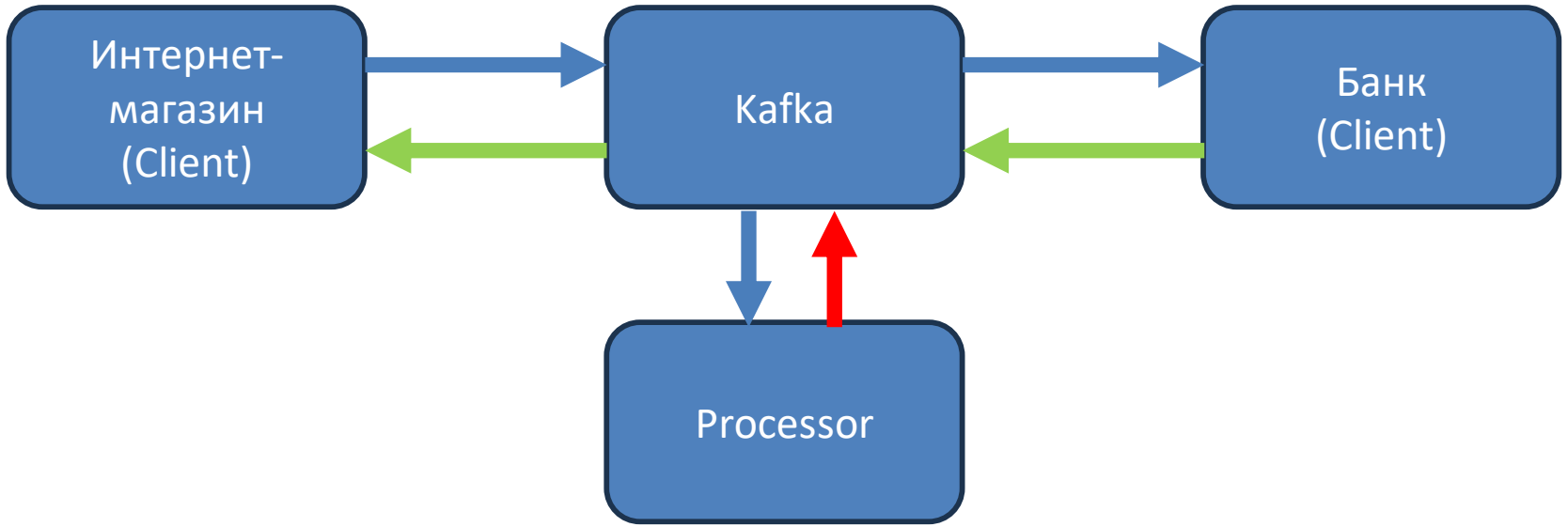
Производители Kafka

Содержание

1. Создание производителей
2. Синхронная/асинхронная отправка сообщений
3. Конфигурирование производителей

Запись сообщений в Kafka

Запись сообщений в Kafka

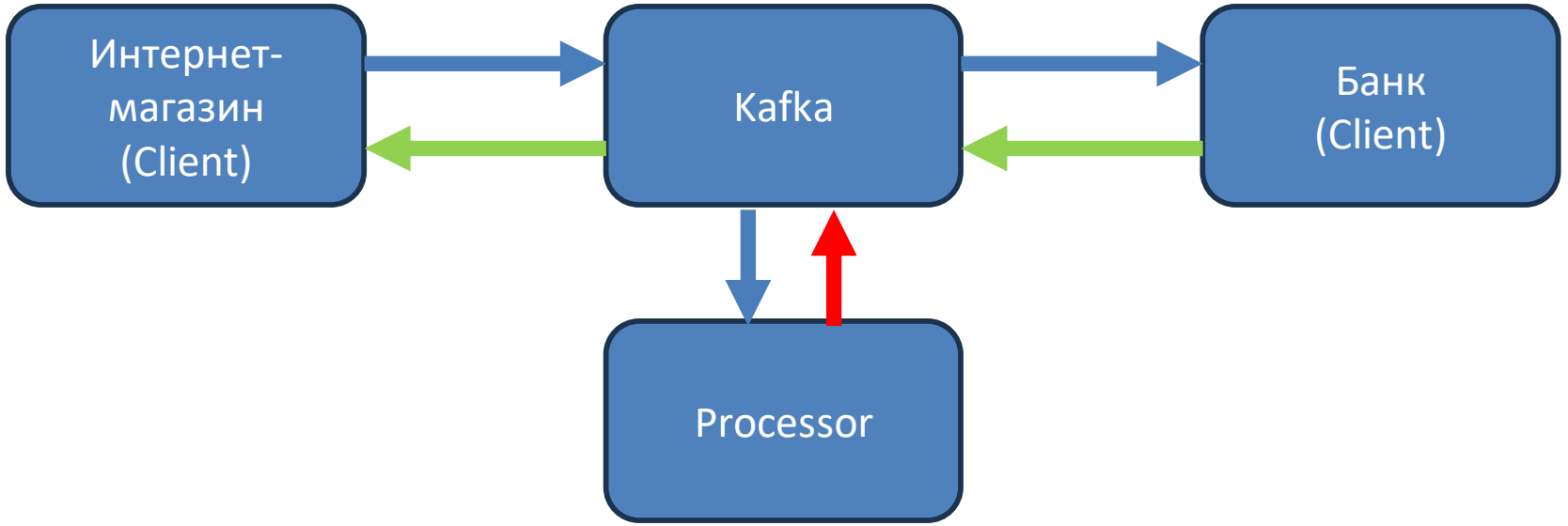


Сторонние производители

- В Kafka имеется двоичный протокол передачи данных;
- Приложения могут читать сообщения из Kafka или записывать сообщения в неё простой отправкой соответствующих последовательностей байтов на сетевой порт Kafka;
- Существует множество клиентов, реализующих протокол передачи данных Kafka на различных языках программирования.

Обзор производителя

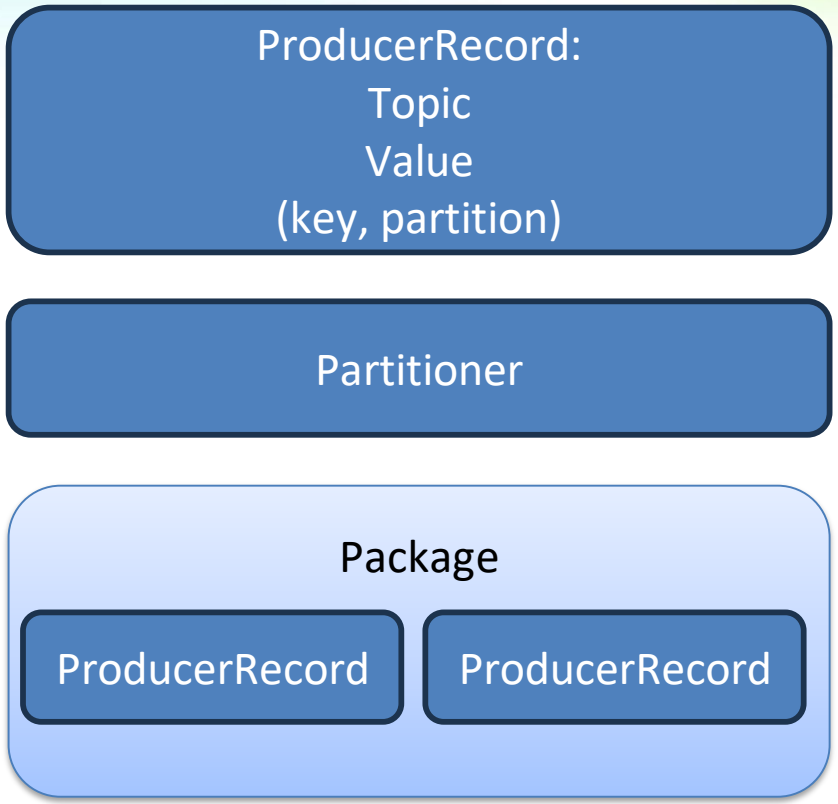
- Фиксация действий пользователей для аудита или анализа;
- Запись показателей;
- Сохранение журнальных сообщений;
- Асинхронное взаимодействие между приложениями;
- Буферизация информации перед записью её в базу данных
- И не только…



Обзор производителя

1. Создание ProducerRecord;
2. Продюсер сериализует значение перед отправкой;
3. Работа объекта Partitioner

   partition == null ? setPartition : continue
4. ProducerRecord помещается в пакет;
5. Поток выполнения отправляет пакет в брокер Kafka;
6. После получения сообщений брокер отправляет ответ;
7. if(success) return RecordMetadata else return error;
8. В случае получения ошибки, производитель может повторить отправку



Создание производителя Kafka

- ✓bootstrap.servers
- ✓key.serializer
- ✓value.serializer

Создание производителя Kafka

```
Properties kafkaProperties = new Properties();// 1

kafkaProperties.put("bootstrap.servers", "localhost:9092");
kafkaProperties.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");// 2
kafkaProperties.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer producer = new KafkaProducer<Stirng,
String>(kafkaProperties);// 3
```

Отправка сообщений в Kafka

- Сделать и забыть
- Синхронная отправка
- Асинхронная отправка

Отправка сообщений в Kafka

```
ProducerRecord<String, String> producerRecord =
        new ProducerRecord<>("CustomerCountry", "Precision Products", "France");// 1

try {
    producer.send(producerRecord);// 2
}
catch (Exception e) {
    e.printStackTrace();// 3
}
```

Синхронная отправка

```
ProducerRecord<String, String> producerRecord =
        new ProducerRecord<>("CustomerCountry", "Precision Products", "France");

try {
    producer.send(producerRecord).get();// 1
}
catch (Exception e) {
    e.printStackTrace();// 2
}
```

Асинхронная отправка сообщения

```
private class DemoProducerCallback implements Callback {// 1

    @Override
    public void onCompletion(RecordMetadata recordMetadata, Exception e) {

        if (e != null) {
            e.printStackTrace();// 2
        }
    }
}

KafkaProducer<String, String> producer = new KafkaProducer<>(new HashMap<>());

ProducerRecord<String, String> producerRecord =
        new ProducerRecord<>("CustomerCountry", "Precision Products", "France");// 3

producer.send(producerRecord, new DemoProducerCallback());// 4
```

Настройка производителей

- acks (0 / 1 / all)
- buffer.memory (block.on.buffer.full, начиная с 0.9.0.0 max.block.ms)
- compression.type (snappy / gzip / lz4)
- retries
- batch.size (объём памяти в байтах, не число сообщений!)
- linger.ms

Настройка производителей

- client.id
- max.in.flight.requests.per.connection
- timeout.ms, request.timeout.ms, metadata.fetch.timeout.ms
- max.block.ms
- max.request.size
- receive.buffer.bytes и send.buffer.bytes

Гарантии упорядоченности

- Apache Kafka сохраняет порядок сообщений внутри партиции
- Если retries=0 и max.in.flights.requests.per.connection > 1, порядок записи сообщений может быть нарушен
- Чтобы гарантировать, что во время повтора попытки не будут отправляться другие сообщения, нужно выставить max.in.flights.requests.per.connection=1

Сериализаторы

Пользовательские сериализаторы

- Avro / Thrift / Protobuf
- Можно создать кастомный сериализатор для любых объектов
- Для создания кастомного сериализатора, необходимо реализовать интерфейс org.apache.kafka.common.serialization.Serializer
- Рекомендуется использовать существующие сериализаторы и десериализаторы

Сериализация с помощью JSON

```
public class CustomJsonSerializer implements Serializer<Customer> {

    private String encoding = StandardCharsets.UTF_8.name();

    @Override
    public byte[] serialize(String topic, Customer data) {

        try {
            return new ObjectMapper().writeValueAsString(data).getBytes(encoding);
        }
        catch (UnsupportedEncodingException e) {
            throw new SerializationException("Error when serializing string to byte[] due to
unsupported encoding " + encoding);
        }
        catch (JsonProcessingException e) {
            throw new SerializationException("Error JSON processing", e);
        }
    }
}
```

Партиции

- Сообщения Kafka представляют собой пары «ключ/ значение»
- На основе ключа определяется, в какую партицию топика записывать сообщение
- Все сообщения с одинаковым ключом попадут в одну партицию

```
ProducerRecord<Integer, String> producerRecord =
        new ProducerRecord<>("CustomerCountry", "Precision Products", "France");
```

```
ProducerRecord<Integer, String> producerRecord =
        new ProducerRecord<>("CustomerCountry", "France");// 1
```

Партиции

- Если ключ равен null и используется метод секционирования по умолчанию, то запись будет отправлена в одну из доступных партиций топика случайным образом
- Kafka вычисляет хеш-значение ключа с помощью собственного алгоритма хеширования
- Соответствие ключей партициям остаётся согласованным лишь до тех пор, пока число партиций в топике не меняется
- Kafka предоставляет возможность реализации пользовательской стратегии секционирования

Пример использования объекта Partitioner

```
public class BananaPartitioner implements Partitioner {

    @Override
    public void configure(Map<String, ?> configs) {}// 1

    @Override
    public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[] valueBytes,
Cluster cluster) {

        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();
        if ((keyBytes == null) || (! (key instanceof String))) {// 2
            throw new InvalidRecordException("We expect all messages to have customer name as key");
        }
        if (((String) key). equals ("Banana")) {
            return numPartitions - 1;
        }

        // Banana всегда попадает в последнюю партицию
        // Другие записи распределяются по партициям путем хеширования
        return (Math.abs(Utils.murmur2(keyBytes)) % (numPartitions - 1));
    }

    @Override
    public void close() {}
}
```

Спасибо за внимание