KARPOV.COURSES >>>
КОНСПЕКТ

> Конспект > 7 урок > Apache Kafka. Spark streaming

> Оглавление

> Message broker

Message broker - архитектурный паттерн в распределённых системах; приложение, которое преобразует сообщение по одному протоколу от приложения-источника в сообщение протокола приложения-приёмника, тем самым выступая между ними посредником.

Паттерн позволяет создать буфер, который может коммуницировать с различными системами по унифицированным протоколам, создавать канал коммуникации между приложениями или системами.

Типы Message broker-ов:

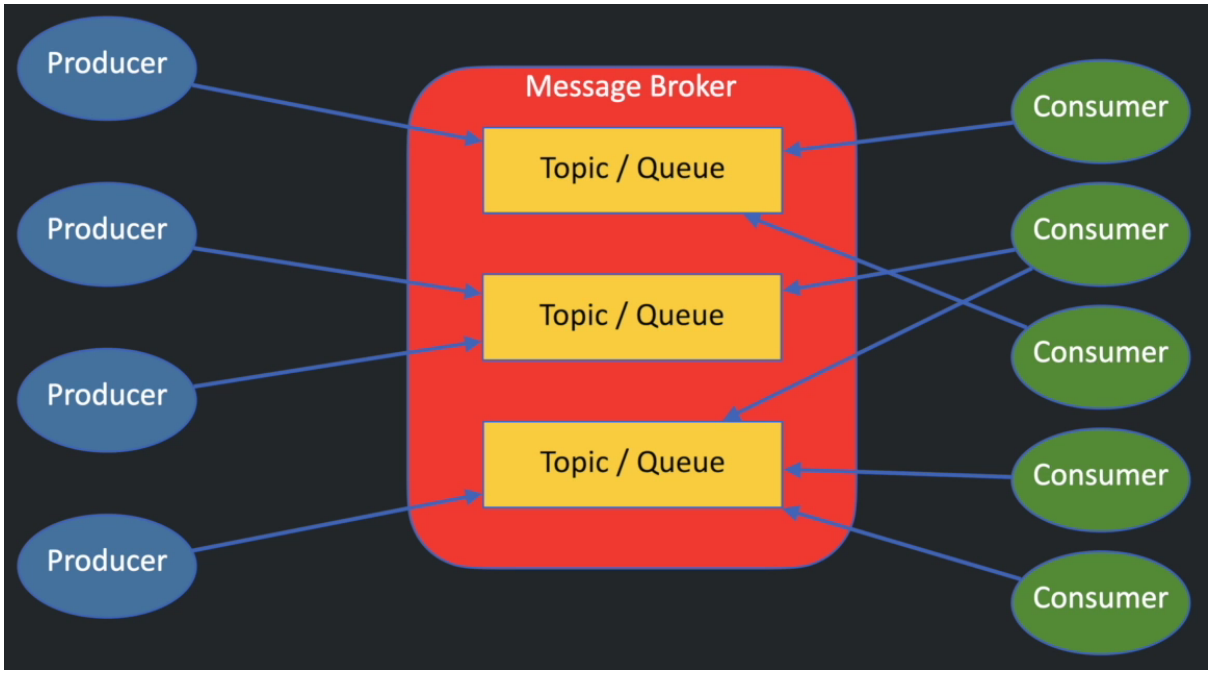
- point-to-point: брокеры, которые работают по принципу доставки конкретного сообщения. Применяется message – ориентированный подход, который основан на гарантированной доставке сообщений и строгой последовательности. Выполняется в виде очереди FIFO, в которую одна система пишет сообщения, а другая система вычитывает эти сообщения
- publish / subscribe: источники (producer-ы) публикуют свои изменения, а потребители (consumer-ы) получают эти сообщения по подписке. Нет гарантии строгой последовательности, но являются более масштабируемыми.

> Назначение Message broker

- интеграция систем с разными протоколами – можем использовать различные языки, т.к. для большинства MB существуют клиентские библиотеки, которые позволяют общаться с MB. Например, можно с помощью MB скоммуницировать одно приложение, написанное на Go, а другое на Java,
- роутинг сообщений – можем настраивать правила отправки сообщений,
- надёжное хранение – при правильной настройке отправленное сообщение не будет потеряно,
- гарантированная доставка – гарантировано, что сообщение будет доставлено, но не все MB гарантируют, что единожды отправленное сообщение не будет получено приёмником несколько раз,
- масштабирование (как источников, так и потребителей) – должны позволять подключать большее количество источников и потребителей,
- преобразование сообщений – сообщение может быть преобразовано внутри MB (например, можно шифровать, маскировать сообщения),

- интеграция с внешними системами – МВ может выступать не просто как канал передачи данных, но и взаимодействовать с внешними системами (например, с системами мониторинга, авторизации), и по результату каким-то образом проверить, обогатить, правильно смаршрутизировать сообщение.

> Общая схема Message broker



Рассмотрим верхнеуровневую схему МВ общего назначения:

Topic / Queue – логический канал передачи информации.

Producer – системы-источники информации подключаются к МВ и пишут свои сообщения в Topic / Queue.

Consumer – системы-получатели либо самостоятельно выбирают информацию из Queue, либо подписываются на изменения данных в Topic.

Несколько Producer-ов могут писать в один Topic / Queue, несколько Consumer-ов могут получать данные из одного Topic / Queue, Consumer может подписаться на несколько Topic / Queue.

> Apache Kafka

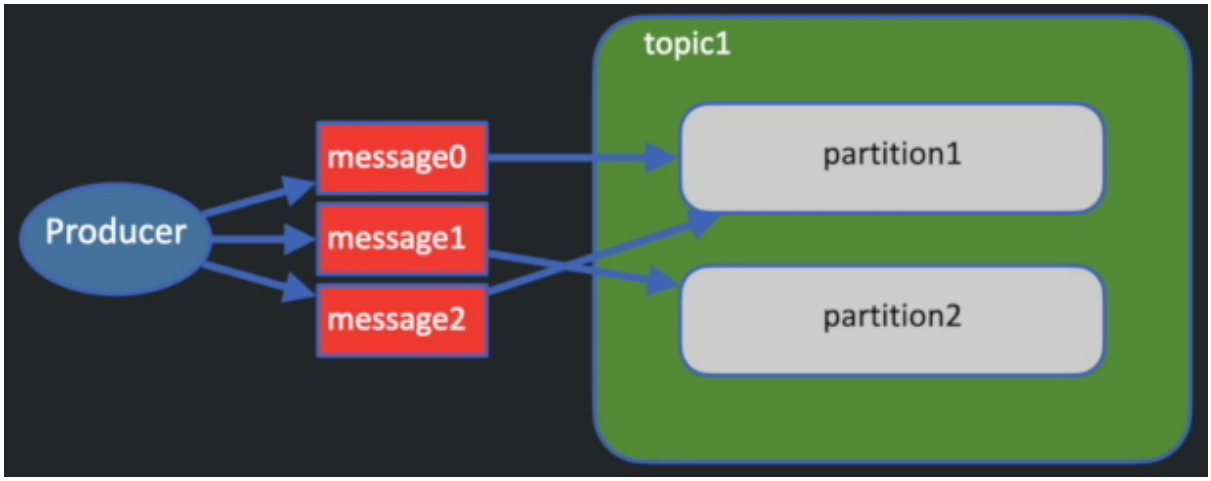
- изначально внутренняя разработка LinkedIn (выложен в opensource в 2011),

- названа в честь Франца Кафки (FranzKafka) потому что «система оптимизирована для записи» и JayKreps(один из главных разработчиков) нравились произведения автора,
- работает по модели publish / subscribe, т.е. публикация изменений и их получение,
- гибко масштабируется: можем не только увеличить количество серверов, но и на существующих серверах можем отмасштабировать логические каналы связи (topic-и),
- репликация: для гарантии того, что данные не будут потеряны даже при выходе из строя какого-либо из серверов,
- данные хранятся в topic,
- сообщения хранятся по структуре key - value + служебный заголовок (header), можем записывать любые сообщения, предварительно переведя их в массив byte, заголовок состоит из нескольких полей и позволяет производить навигацию внутри topic,
- навигация по topic-у производится на основе offset (смещения) – инкрементальный идентификатор события внутри каждой partition (партиции), все сообщения нумеруются, и мы получаем сообщения в том порядке, в котором они были пронумерованы. Offset не является глобальной величиной для topic-а, а работает на уровне части topic-а, которая называется partition (партиция). Если topic состоит из более чем одной партиции, то нет гарантии строгой последовательности получения сообщений,
- данные хранятся согласно Log retention и Cleanup policy. Log retention – правила, по которым Kafka принимает решение о том, что данные устарели, и можно проводить чистку этих данных. Cleanup policy – правила о действиях при наступлении какого-либо события.

> Внутренняя иерархия Apache Kafka

Producer подключается к MB и говорит в какой topic он хочет писать, и если такого topic не существует, то:

- он создаётся автоматически, если включена соответствующая опция,
- запрашиваем у администратора создание нужного нам topic-а с выдачей на него прав доступа и правил записи/чтения.



Topic:

- логическая append-only очередь сообщений (message): можем только добавлять сообщения, а удалить ошибочное сообщение не можем,
- состоит из 1+ партиций (partition): для максимального распараллеливания обработки сообщений как с точки зрения записи, так и чтения.

Partition:

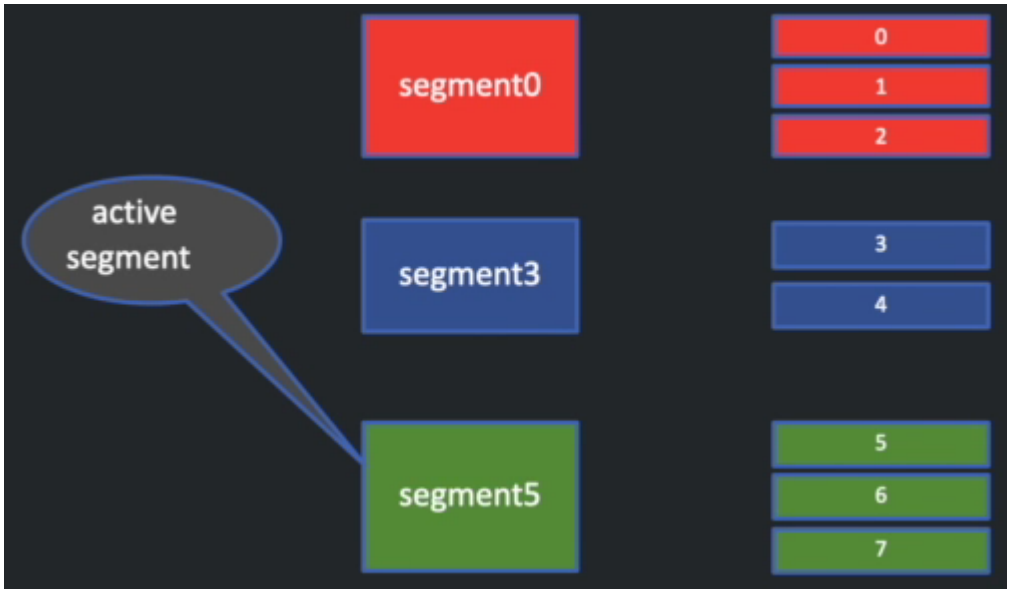
- физическая единица хранения данных topic: состоит из файлов, которые хранятся на сервере,
- в partitionможно писать, но нельзя удалять,
- в конкретный момент времени жёстко привязана к конкретному broker-у. Есть репликация, но только одна partition будет active, остальные follower,
- запись и чтение из partition:

- если используется ключ для сообщения, то на основании индекса, который является остатком от целочисленного деления hash от ключа на количество партиций, определяется в какую partition сообщение будет отправлено. Consumer, подключаясь к topic-у, понимает топологию внутри topic-а и подключается к одной или нескольким partition. Consumer-ов можно объединять в группы, чтобы сообщение было получено только один раз, но в рамках группы.

- если не используется ключ для сообщения (в большинстве случаев), то распределение между partition происходит по принципу round-robin – каждое следующее сообщение пишется в следующую partition и далее по кругу.

Segment:

- набор записей внутри partition,
- физически являются файлами в файловой системе,
- всегда есть один active segment, в который идёт запись новых message,
- при превышении размера сегмента создаётся новый, который становится активным.



Message состоит из:

- заголовок, в котором хранятся:

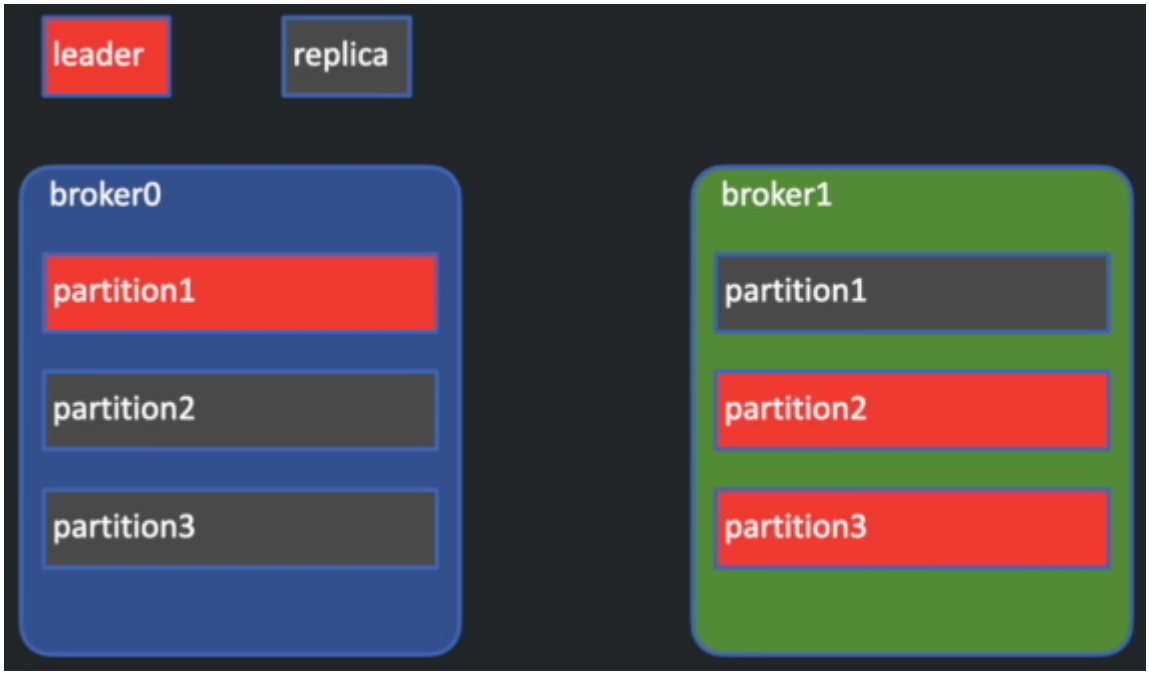
1. поля topic (название топика) + partition (партиция, из которой сообщение было вычитано) + offset (смещение внутри партиции) + timestamp (когда сообщение было доставлено в Kafka),
2. additional headers: дополнительные заголовки со структурой словаря со строками (Map[String, String]),

- body: тело сообщения:

1. key – byte[] (ключ, который является массивом byte),
2. value – byte[] (значение, которое является массивом byte)

Broker – каждый из серверов в Kafka:

- обслуживает topic partitions
- возможна репликация (leader + followers), можем иметь несколько копий партиций в каждом топике.



> Kafka Log retention, Cleanup policy

Log retention – правила, которые позволяют избавляться от устаревших данных. Для каждого topic-а можем установить свою политику log retention.

- time based: на основе времени, по истечении срока маркирует данные (по умолчанию 7 дней). Настройки в порядке убывания приоритета:

log.retention.ms

log.retention.minutes

log.retention.hours

- size based: на основе размера, какой максимальный объём можем хранить в topic-е:

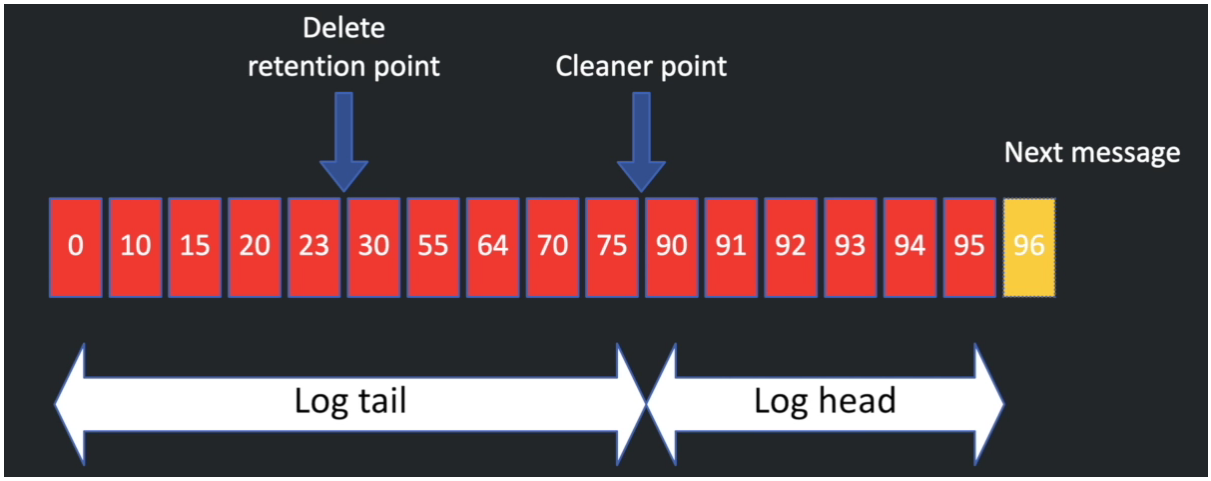
log.retention.size

Cleanup policy – правила, по которым производится очистка данных.

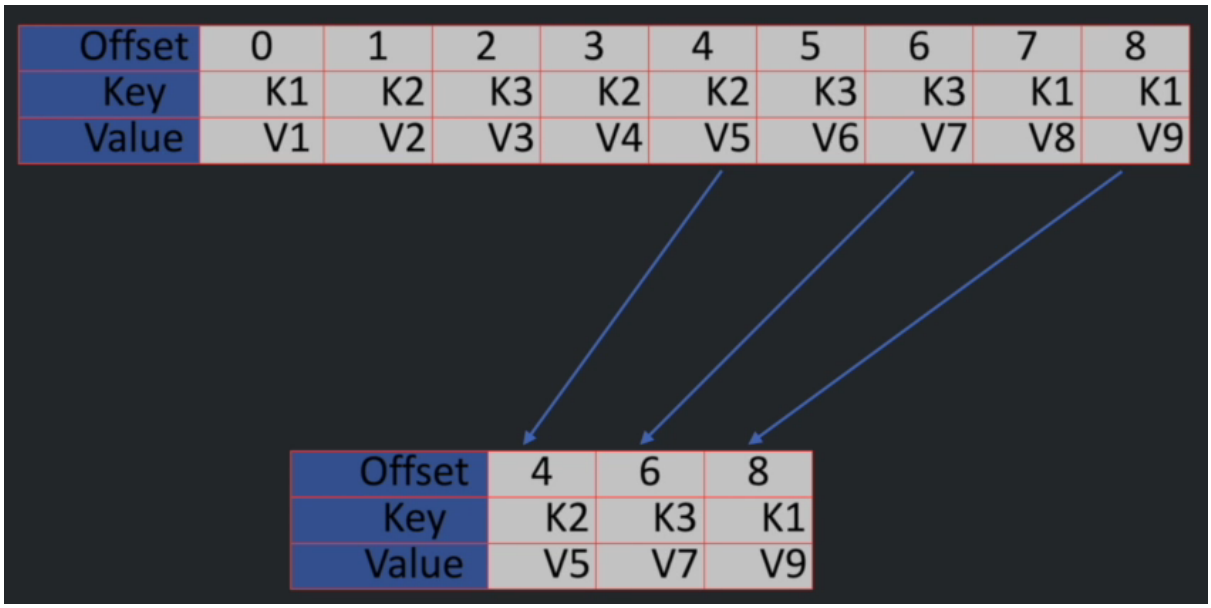
- DELETE (по умолчанию): помечает на удаление segment при устаревании / превышении размера, и далее асинхронно он удаляется,
- COMPACT: производится фильтрация событий, оставляет только последние сообщения (message) для каждого ключа (message key),
- DELETEAND COMPACT:

1. производится compaction,
2. удаление согласно retention policy

> Cleanup policy compaction



В topic-е есть partition,а в них лежат сообщения. В partition есть log tail – хвост сообщений, в котором удалены из последовательности некоторые сообщения, log head – монотонная возрастающая по номеру последовательность сообщений. Указатель cleaner point указывает на начало log head.



Берём все сообщения, далее:

1. сбор статистики в log head по каждому ключу: выбираются самые актуальные записи (last offset в log head),
2. Log recopy с удалением старых записей с одинаковым ключом (требуется дополнительное место на диске): копирует только актуальные записи, т.е. только последние записи по каждому ключу, и формируется новый лог без повторений по ключу,
3. Swap (переключение) нового и старого лога, операция является атомарной.

> Spark streaming

Spark streaming и другие системы обработки потоковой информации в больших данных относятся к системам near real time, т.е. имеют свою близость к действительному real time.

Системы обработки потоковой информации делятся на те, которые:

1. Оперируют понятием сообщение как единицей информации.
2. Получает информацию из источника micro-batch-ами.

Spark streaming:

- оперирует понятием micro-batch,
- Discretized Streams (DStreams):

1. появился в Spark 0.7.0,
2. RDD-based batch, на каждый micro-batch создаётся RDD,
3. считается устаревшим

- Structure streaming:

1. появился в Spark 2.0 (stable в 2.2),
2. DataFrame API: позволяет смешивать классические batch-обработки и обработки потоковой информации, используя один и тот же API, т.е. неважно, откуда этот DataFrame пришёл,
3. catalyst,
4. постоянно развивается

Главное отличие Structure streaming от Discretized Streams заключается в том, что полученную информацию представляем как большую «бесконечную»

таблицу, которую мы можем обрабатывать.

> Structure streaming

- Концепция «бесконечной» таблицы.
- Поддержка watermarks (метки для использования данных прошлых micro-batch).
- DataFrame API (streaming join static): можем комбинировать DataFrame API как стриминговых, так и статичных.
- Fault tolerance:

1. checkpoint: информация о последних обработанных offset per partition per topic, которая складывается в папку, указанную на момент запуска приложения;
2. несколько политик для trigger: определяют время обработки потоковых данных, будет ли запрос выполняться как micro-batch запрос с фиксированным интервалом batch обработки или как запрос непрерывной обработки;
3. динамические метрики: можем получать информацию о том, что сейчас происходит в поточной обработке.

> Structure streaming source, sink, triggers

> Structure streaming source

Есть источник и приёмник и между ними выстраиваем логику взаимодействия.

По типу представления источники делятся на категории:

- File (файловый источник): text, CSV, JSON, ORC, Parquet,
- Kafka,
- Socket – чтение данных из сокета (обычно для отладки), подключение по IP-адресу к порту,
- Rate – для генерации данных (тестирование + benchmarking), генерирует случайным образом нужное количество сообщений в секунду.

> Structure streaming sink

По типу представления приёмники делятся на категории:

- File (файловый приёмник),
- Kafka,
- Foreach (foreach & foreachBatch) – позволяет применять операции каждому row / batch. Также позволяет несколько раз переиспользовать данные и писать в нескольких output (что нельзя делать для других типов sink),
- Console – для отладки и тестирования, вывод информации в консоль,
- Memory – хранит данные в памяти (table_name == query_id) – обычно для отладки и демо-примеров.

> Structure streaming triggers

Trigger – правило срабатывания micro-batch-а.

- Unspecified (default) – запуск micro-batch по готовности:

spark.streaming.receiver.maxRate

spark.streaming.kafka.maxRatePerPartition

- Fixed interval – запуск через равные промежутки времени:

предыдущий успел выполниться – ждёт конца интервала,

предыдущий не успел выполниться – ждёт окончания и сразу запускается,

нет данных – не запускается.

- One-time – запускается один раз, получает все доступные данные и обрабатывает, после чего останавливается (для небольшого количества информации).
- Continuous with fixed checkpoint interval – экспериментальный – позволяет обрабатывать данные с малой задержкой (~1 ms).

> Глоссарий

JMX (Java Management Extensions) – технология Java, предназначенная для контроля и управления приложениями, системными объектами, устройствами и компьютерными сетями.

Topic – логический именованный канал в Message broker, где хранятся и организованы сообщения, используется когда один или несколько источников (producer) хотят разослать сообщение группе приёмников (consumer), которые подписаны на тот или иной topic.

Queue – логический именованный канал в Message broker, где хранятся и организованы сообщения, используется в сценарии, когда один или несколько источников (producer) хотят отправить сообщение одному приёмнику (consumer). Сообщения поступают в очередь по порядку, а приёмник получает сообщения из очереди одно за другим.

Micro-batch – множество сообщений, которые формируются по каким-либо условиям и являются атомарными единицами чтения и записи информации в Spark streaming.

API (application programming interface) – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.