Краткое оглавление

Содержание

Предисловие

Начиная с первого выпуска, вышедшего в 2011 году, технологии Apache Kafka® помогли создать новую категорию систем передачи данных, и теперь они являются основой бесчисленного множества современных приложений, управляемых событиями. В своей книге «Kafka в действии» Дилан Скотт (Dylan Scott), Виктор Гамов (Viktor Gamov) и Дейв Клейн (Dave Klein) делятся навыками проектирования и реализации приложений на основе событий, реализованных с использованием Apache Kafka. Авторы имеют богатый опыт работы с Kafka в реальном мире, что выделяет эту книгу среди других.

Давайте на минутку зададимся вопросом: «Зачем вообще нужна платформа Kafka?» Исторически сложилось так, что большинство приложений были основаны на системах хранения данных. Когда в мире происходили какие-то интересные события, они немедленно сохранялись в этих системах, но реакция на эти события происходила позже – либо когда пользователь явно запрашивал информацию, либо в ходе выполнения некоторых заданий пакетной обработки.

В системах передачи данных приложения строятся путем предварительного определения того, что они должны делать при появлении новых событий. Когда случаются новые события, приложения автоматически реагируют на них практически мгновенно. Такие приложения, управляемые событиями, привлекательны тем, что позволяют предприятиям гораздо быстрее извлекать новую информацию из своих данных. Однако переход к приложениям, управляемым событиями, требует изменения мышления, что не всегда легко. Эта книга предлагает исчерпывающее описание событийно-ориентированного мышления, а также реалистичные практические примеры, которые вы сможете опробовать.

«Kafka в действии» объясняет, как работает Kafka, и особое внимание уделяет созданию комплексных приложений, управляемых событиями, на основе Kafka. Здесь вы познакомитесь с компонентами, необходимыми для создания простого приложения Kafka, а также узнаете, как создавать сложные приложения с использованием таких библиотек, как Kafka Streams и ksqlDB. Также в этой книге рассказывается, как после создания приложения развернуть его в промышленном окружении, и освещаются такие ключевые темы, как мониторинг и безопасность.

Я надеюсь, что вам понравится эта книга так же, как мне. Удачной передачи событий!

– Юн Рао (Jun Rao), соучредитель Confluent

Вступление

Один из вопросов, который часто задают нам, когда мы рассказываем о работе над технической книгой: почему был выбран именно формат печатной книги? Дилан, например, всегда предпочитал узнавать что-то новое, читая книги. Другой фактор – ностальгия, навеваемая воспоминаниями о первой технической книге по программированию, которую он прочитал, «Elements of Programming with Perl» Эндрю Л. Джонсона (Andrew L. Johnson), выпущенной издательством Manning в 2000 году. Эта книга особенно запомнилась ему, и он до сих пор вспоминает, насколько приятно было читать ее страницы. Мы надеемся доставить такое же удовольствие своим читателям, описывая Apache Kafka.

Предвкушение познания чего-то нового почувствовал каждый из нас, когда мы впервые начали работать с Kafka. На наш взгляд, Kafka существенно отличается от любых других брокеров сообщений или шин служб предприятия (Enterprise Service Bus, ESB), которые нам доводилось использовать раньше. Быстрота разработки производителей и потребителей сообщений, возможность повторной обработки данных и скорость, с которой независимые потребители перемещаются без удаления данных из других потребительских приложений, позволяют решать проблемы, встречавшиеся в прошлом, и впечатлили нас больше всего, когда мы стали изучать возможность применения Kafka.

Мы видим, что Kafka меняет стандарты для платформ данных; она способна помочь перенести пакетные рабочие процессы и рабочие процессы извлечения, преобразования и загрузки (Extract, Transform, Load, ETL) ближе во времени к потокам данных. Поскольку эта платформа отходит от архитектур обработки данных, использовавшихся раньше и известных многим корпоративным пользователям, мы хотели помочь пользователям, не знакомым с Kafka, научиться работать с производителями и потребителями Kafka, а также решать базовые задачи разработки и администрирования Kafka. Мы надеемся, что к концу этой книги вы почувствуете в себе готовность углубиться в исследование более сложных тем Kafka, таких как мониторинг кластеров, создание метрик и межсайтовая репликация данных.

Всегда помните, что эта книга запечатлела момент, как Кафка выглядит сегодня. Она почти наверняка будет меняться и, надеюсь, станет еще лучше к тому времени, когда вы будете читать эту работу. Мы верим, что эта книга направит вас на увлекательный путь изучения основ Apache Kafka.

Об этой книге

Мы писали «Kafka в действии» как практическое руководство по началу работы с Apache Kafka. В этой книге читатели встретят небольшие примеры, объясняющие некоторые параметры и настройки, которые можно использовать для изменения поведения Kafka в соответствии с конкретными вариантами применения. Ядро Kafka специально создавалось как настраиваемое в широких пределах и легко интегрирующееся с другими продуктами, такими как Kafka Streams и ksqlDB. Мы надеемся показать, как можно использовать платформу Kafka для удовлетворения различных бизнес-требований, чтобы вы освоились с ней к концу этой книги и знали, с чего начать решение ваших задач.

Кому адресована эта книга

«Kafka в действии» адресована разработчикам, желающим познакомиться с идеей потоковой обработки данных. От читателя не требуется обладать какими-либо знаниями о Kafka, но базовые знания командной строки и умение ею пользоваться не будут лишними. В Kafka есть несколько мощных инструментов командной строки, которые мы используем, и пользователь должен уметь по крайней мере вводить команды в командной строке.

Также могут пригодиться некоторые навыки программирования на языке Java и способность распознавать идеи программирования на любом языке. Эти навыки помогут понять представленные примеры кода, которые реализованы в основном в стиле Java 11 (а также Java 8). Кроме того, хотя это и необязательно, будет полезно общее понимание архитектуры распределенных приложений. Чем больше пользователь знает о репликациях и сбоях, тем проще ему будет понять, например, как Kafka использует реплики.

Организация книги

Эта книга состоит из трех частей, разбитых на 12 глав. Часть I представляет ментальную модель Kafka и рассказывает, где может пригодиться Kafka в реальном мире:

- глава 1 содержит общее введение в Kafka, опровергает некоторые мифы и описывает примеры использования в реальных условиях;
- глава 2 исследует архитектуру Kafka в общих чертах и вводит важную терминологию.

Часть II переходит к основным компонентам Kafka – клиентам и самому кластеру:

- глава 3 рассматривает характеристики проектов, в которых с успехом можно было бы использовать Kafka, и описывает некоторые подходы к разработке новых проектов. Здесь также обсуждается необходимость схем, на которые следует обратить особое внимание на этапе создания проекта на основе Kafka, но не позже;
- глава 4 иллюстрирует некоторые детали создания клиента-производителя и параметры управления передачей ваших данных в кластер Kafka;
- глава 5 меняет фокус главы 4 и иллюстрирует приемы получения данных из Kafka с помощью клиента-потребителя. Здесь будет представлена идея смещений и повторной обработки данных, обусловленная возможностью хранения сообщений;
- глава 6 рассматривает роль брокеров в кластере и как они взаимодействуют с вашими клиентами. Здесь мы исследуем различные компоненты, такие как контроллер и реплика;
- глава 7 обсуждает понятия тем и разделов, включая возможность компактификации тем и особенности хранения разделов;
- глава 8 описывает инструменты и архитектуры для обработки данных, которые может потребоваться сохранить или обработать повторно. Необходимость хранения данных в течение нескольких месяцев или лет может привести к тому, что вам придется подумать о вариантах хранения за пределами кластера;
- глава 9 завершает часть II обзором журналов, метрик и административных функций, помогающих поддерживать работоспособность кластера.

В части III мы перейдем от знакомства с основными компонентами частей Kafka к изучению вариантов улучшения работающего кластера:

- глава 10 представляет варианты усиления защиты кластера Kafka с помощью SSL, списков управления доступом ACL и квот;
- глава 11 посвящена реестру схем Schema Registry и особенностям его использования для работы с данными и сохранения совместимости с предыдущими и будущими версиями наборов данных. Считается, что эта возможность предназначена для приложений корпоративного уровня, однако она с успехом может использоваться для обслуживания любых данных, изменяющихся с течением времени;

- глава 12, последняя, посвящена знакомству с Kafka Streams и ksqlDB. Эти продукты относятся к более высоким уровням абстракции и основаны на ядре, которому посвящена вся вторая часть книги. Kafka Streams и ksqlDB – достаточно обширные темы, поэтому в нашем введении мы представим ровно столько информации, сколько действительно необходимо, чтобы приступить к самостоятельному изучению этих продуктов.

О примерах программного кода

Эта книга содержит множество примеров исходного кода как в пронумерованных листингах, так и в обычном тексте. В обоих случаях исходный код `оформлен моноширинным шрифтом`, чтобы визуально отделить его от обычного текста. Во многих случаях исходный код был дополнительно отформатирован; мы добавили разрывы строк и изменили отступы, чтобы уместить примеры по ширине книжной страницы. В некоторых случаях даже этого оказалось недостаточно, и в каких-то листингах вы можете встретить символы ➥, обозначающие продолжение строк. Многие листинги сопровождаются дополнительными комментариями, поясняющими важные понятия.

Наконец, следует отметить, что многие примеры кода не предназначены для выполнения в форме самостоятельных программ, – это выдержки, иллюстрирующие наиболее важные стороны обсуждаемого. Все примеры из книги и сопровождающий их исходный код в полной форме вы найдете на GitHub по адресу https://github.com/Kafka-In-Action-Book/Kafka-In-Action-Source-Code и на сайте издателя www.manning.com/books/kafka-in-action. Также выполняемые фрагменты кода можно получить в онлайн-версии этой книги по адресу https://livebook.manning.com/book/kafka-in-action.

Живое обсуждение книги

Приобретая книгу «Kafka в действии», вы получаете бесплатный доступ к частному веб-форуму, организованному издательством Manning Publications, где можно оставлять комментарии о книге, задавать технические вопросы, а также получать помощь от автора и других пользователей. Чтобы иметь доступ к форуму и зарегистрироваться на нем, откройте в веб-браузере страницу https://livebook.manning.com/#!/book/kafka-in-action/discussion. Узнать больше о форумах Manning и познакомиться с правилами поведения можно по адресу https://livebook.manning.com/#!/discussion.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны авторов отсутствуют какие-либо обязательства уделять фо-

руму какое-то определенное внимание – их присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать авторам стимулирующие вопросы, чтобы их интерес не угасал! Форум и архив с предыдущими обсуждениями остается доступным на сайте издательства, пока книга продолжает издаваться.

Другие онлайн-ресурсы

Все изменения, происходящие в Kafka с течением времени, неизменно отражаются перечисленными ниже ресурсами. В большинстве случаев на этих сайтах можно найти документацию с описанием прошлых версий:

- документация по Apache Kafka – http://kafka.apache.org/documentation.html;
- документация Confluent – https://docs.confluent.io/current;
- портал разработчиков Confluent – https://developer.confluent.io.

Об авторах

Дилан Скотт (Dylan Scott) – разработчик программного обеспечения с более чем десятилетним опытом программирования на Java и Perl. После знакомства с системой обмена сообщениями Kafka как средством передачи больших объемов данных Дилан начал погружаться в мир Kafka и технологий потоковой обработки. Он имеет значительный опыт использования таких технологий и очередей, как Mule, RabbitMQ, MQSeries и Kafka.

Дилан обладает различными сертификатами, свидетельствующими об опыте работы в отрасли: PMP, ITIL, CSM, Sun Java SE 1.6, Oracle Web EE 6, Neo4j и Jenkins Engineer.

Виктор Гамов (Viktor Gamov) – пропагандист передовых практик разработки в Confluent, компании, разрабатывающей платформу потоковой передачи событий на основе Apache Kafka. За свою долгую карьеру Виктор накопил богатый опыт в сфере разработки архитектур корпоративных приложений с использованием технологий с открытым исходным кодом. Ему нравится помогать архитекторам и разработчикам проектировать и создавать масштабируемые и высокодоступные распределенные системы с малым временем реакции.

Виктор является профессиональным докладчиком на конференциях по темам распределенных систем, потоковой передачи данных, JVM и DevOps, а также регулярно посещает мероприятия, такие как JavaOne, Devoxx, OSCON, QCon и др. Является соавтором книги «Enterprise Web Development» (O'Reilly Media, Inc.).

Следуйте за Виктором в Твиттере *@gamussa*, где он пишет о спортивной жизни, вкусной и здоровой пище, открытом исходном коде и, конечно же, о Kafka!

Дейв Клейн (Dave Klein) имеет 28-летний опыт работы разработчиком, архитектором, руководителем проекта, автором, преподавателем, организатором конференций и семейного учителя, пока недавно не получил работу своей мечты пропагандиста передовых практик разработки в Confluent. Дейв восхищается удивительным миром потоковой передачи событий Apache Kafka и всеми силами старается помочь другим исследовать его.

Часть I

Начало

В первой части этой книги мы познакомим вас с Apache Kafka и начнем рассматривать реальные случаи использования Kafka:

- в главе 1 подробно опишем преимущества Kafka и развеем некоторые мифы, которые вы, возможно, слышали о Kafka в связи с Hadoop;
- в главе 2 мы сосредоточим наше внимание на высокоуровневой архитектуре Kafka, а также на некоторых других компонентах, составляющих экосистему Kafka: Kafka Streams, Connect и ksqlDB.

К концу этой части вы будете готовы начать принимать и отправлять сообщения в Kafka. Надеемся, что вы также усвоите некоторые ключевые термины.

1 Введение в Kafka

Эта глава охватывает следующие темы:

- преимущества Kafka;
- распространенные мифы о больших данных и системах сообщений;
- реальные примеры использования, когда технологии Kafka помогли улучшить обмен сообщениями, потоковую передачу и обработку данных IoT.

Многие разработчики постоянно сталкиваются с миром, наполненным данными, то и дело льющимися со всех сторон, и нередко оказываются перед фактом, когда устаревшие системы тормозят движение вперед. Одним из основных элементов новых инфраструктур данных, занявших лидирующие позиции в ИТ-ландшафте, является Apache Kafka®[1]. Kafka меняет стандарты платформ данных. Она подталкивает к переходу от извлечения, преобразования и загрузки (Extract, Transform, Load, ETL) данных с использованием пакетных процессов (которые обычно запускаются в одно и то же время) к потокам данных, действующим практически в режиме реального времени [1]. Пакетная обработка, которая когда-то была типичной рабочей лошадкой в сфере обработки корпоративных данных, является, пожалуй, не тем, к чему захочется вернуться по-

[1] Apache, Apache Kafka и Kafka являются товарными знаками Apache Software Foundation.

сле знакомства с мощным набором возможностей, предлагаемых Kafka. На самом деле вы просто не сможете справиться с растущим снежным комом поступающих данных, если не возьмете на вооружение что-то новое.

С таким большим количеством данных системы будут постоянно перегружены работой. Устаревшие системы могут не справиться с обработкой данных в ночной период и продолжат выполнение пакетных заданий на следующий день. Чтобы не отставать от этого постоянного потока данных, обработка информации должна производиться по мере ее поступления – это единственная возможность обеспечить актуальное состояние системы.

Kafka следует многим новейшим и наиболее практичным тенденциям в современном мире информационных технологий и упрощает повседневную работу. Например, Kafka уже нашла свое место в архитектуре микросервисов и интернете вещей (Internet of Things, IoT). Как технология, принятая на вооружение большим числом компаний, Kafka предназначена не только для фанатов или охотников за альфа-версиями. Давайте начнем знакомство с Kafka с общего обзора этой платформы и некоторых особенностей современных потоковых платформ.

1.1. Что такое Kafka?

На сайте проекта Apache Kafka (http://kafka.apache.org/intro) Kafka определяется как распределенная потоковая платформа, предлагающая три основные возможности:

- чтение сообщений из очереди и запись их в очередь;
- надежное хранение сообщений;
- обработку потоков данных по мере их появления [2].

Читателям, не сталкивающимся в своей практике с очередями или брокерами сообщений, может понадобиться помощь, когда мы начнем обсуждать общее назначение и принцип действия такой системы. В общем случае Kafka можно рассматривать как ИТ-эквивалент ресивера (приемника), встроенного в домашний кинотеатр. На рис. 1.1 показан поток данных между ресивером и конечными потребителями.

Как можно видеть на рис. 1.1, цифровые спутниковые, кабельные и Blu-ray™-проигрыватели могут подключаться к центральному приемнику. Эти отдельные компоненты можно рассматривать как источники данных, использующие известный им формат. Во время воспроизведения фильма или компакт-диска они порождают непрерывный поток данных. Ресивер обрабатывает этот поток и преобразует его в формат, понятный внешним устройствам, подключенным к другому концу (ресивер отправляет видеопоток

на телевизор, а звук – на декодер и на динамики). Но какое отношение это имеет к Kafka? Давайте посмотрим на те же потоки данных с точки зрения Kafka (рис. 1.2).

Kafka служит интерфейсом, обеспечивающим возможность взаимодействий клиентов с другими системам. Одни клиенты, которые называются *производителями*, отправляют потоки данных брокерам Kafka. Брокеры выполняют ту же функцию, что и ресивер на рис. 1.1. Другие клиенты Kafka называются *потребителями* – они могут читать данные из брокеров и обрабатывать их. Одни и те же данные может получать не один потребитель. Производители и потребители полностью отделены друг от друга, что позволяет каждому клиенту работать независимо. Подробнее о том, как это делается, вы узнаете в последующих главах.

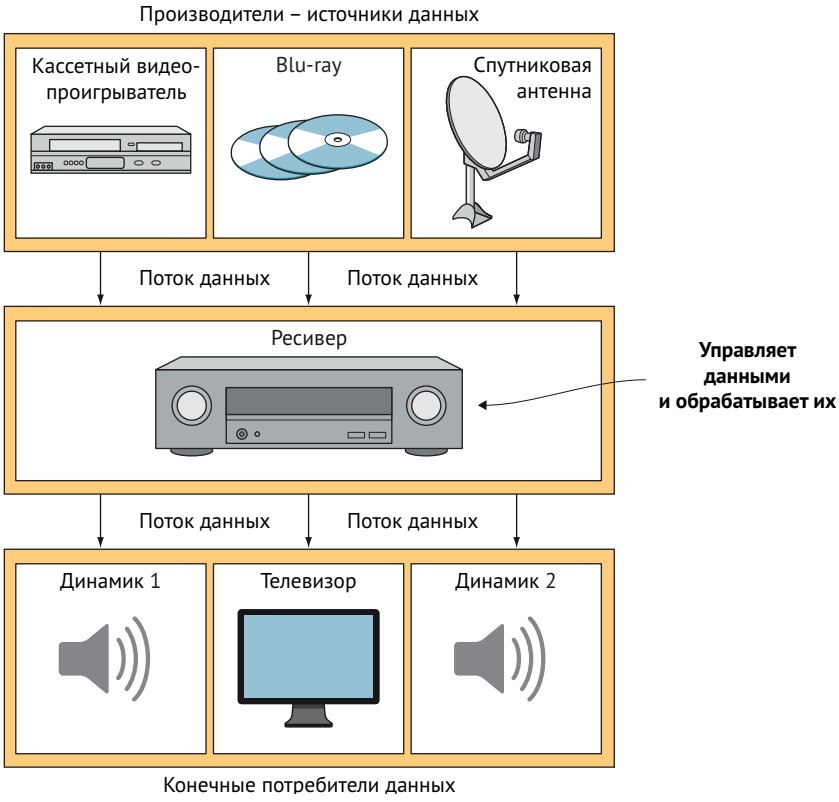

Рис. 1.1. Производители, потребители и потоки данных

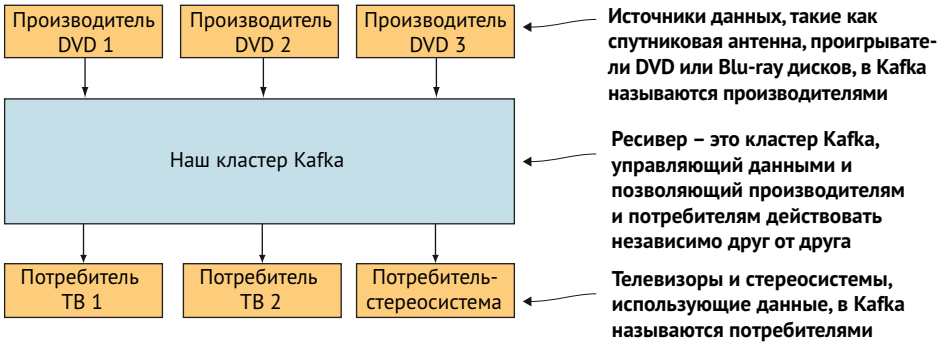

Рис. 1.2. Место Kafka в потоке данных между производителями и потребителями

Как и другие платформы обмена сообщениями, Kafka действует (если говорить упрощенно) как посредник, передавая данные, поступающие в систему (от производителей) и исходящие из системы (к потребителям или конечным пользователям). Такое отделение производителей и потребителей друг от друга позволяет обеспечить минимальную взаимозависимость (слабую связанность) между ними. Производитель может отправить любое сообщение, даже не имея ни малейшего представления о том, ожидает ли эти сообщения хоть кто-нибудь. Кроме того, Kafka поддерживает разные способы доставки сообщений, соответствующие разным бизнес-потребностям. Доставка сообщений в Kafka может осуществляться как минимум тремя способами [3]:

- *не менее одного раза* (at-least-once) – сообщение будет отправляется потребителям до тех пор, пока те не подтвердят его получение;
- *не более одного раза* (at-most-once) – сообщение отправляется только один раз и в случае сбоя не отправляется повторно;
- *точно один раз* (exactly-once) – потребитель гарантированно получит сообщение ровно один раз.

Давайте разберемся, что означают эти варианты обмена сообщениями. Рассмотрим первой семантику «не менее одного раза» (рис. 1.3). Kafka можно настроить так, чтобы она позволяла производителям отправлять одно и то же сообщение многократно и передавала их брокерам. Если производитель не получил подтверждения передачи сообщения брокеру, то он может отправить сообщение повторно [3]. В случаях, когда потеря сообщений недопустима, скажем, сообщений об оплате счета, эта семантика может потребовать дополнительной фильтрации на стороне потребителя, зато это один из самых надежных способов доставки.

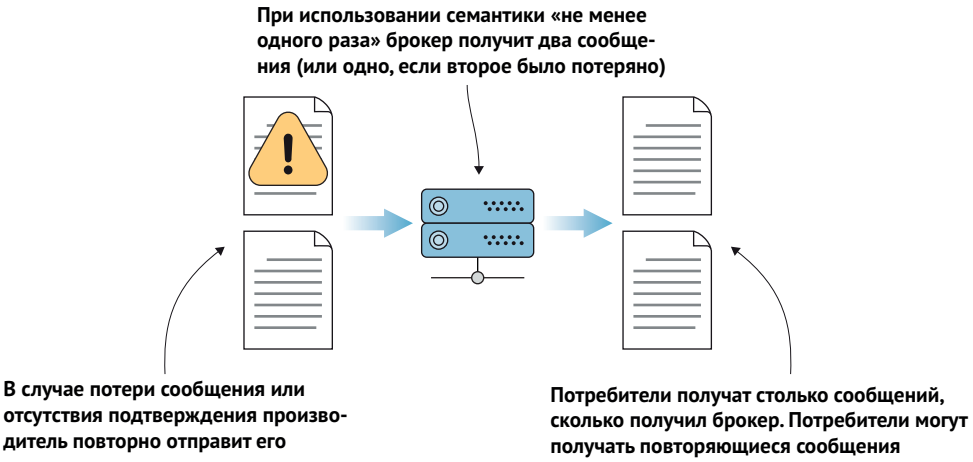

Рис. 1.3. Поток сообщений с использованием семантики «не менее одного раза»

При использовании семантики «не более одного раза» (рис. 1.4) производитель отправляет сообщение только один раз и никогда не повторяет попытку. В случае сбоя производитель движется дальше, не пытаясь отправить сообщение [3]. Но разве потеря сообщений может быть допустима? Представьте популярный веб-сайт, отслеживающий количество просмотров страниц. Для такого сайта вполне возможно потерять несколько событий просмотра из миллионов, которые он обрабатывает каждый день. Увеличение производительности за счет отказа от ожидания подтверждений может перевесить любые потери данных.

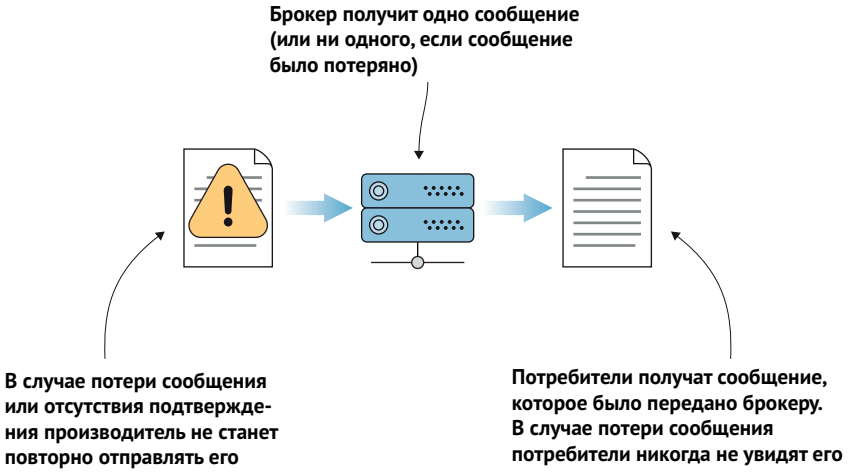

Рис. 1.4. Поток сообщений при использовании семантики «не более одного раза»

Наконец, в версии Kafka 0.11.0 появилась поддержка семантики «точно один раз» (Exactly-Once Semantic, EOS). Появление поддержки этой семантики вызвало множество споров [3]. С одной стороны, семантика «точно один раз» (рис. 1.5) идеально подходит для многих случаев использования. Она выглядит как логическая гарантия удаления дубликатов сообщений. Но большинству разработчиков больше по душе другая логика: отправка одного сообщения и получение того же сообщения на стороне потребителя.

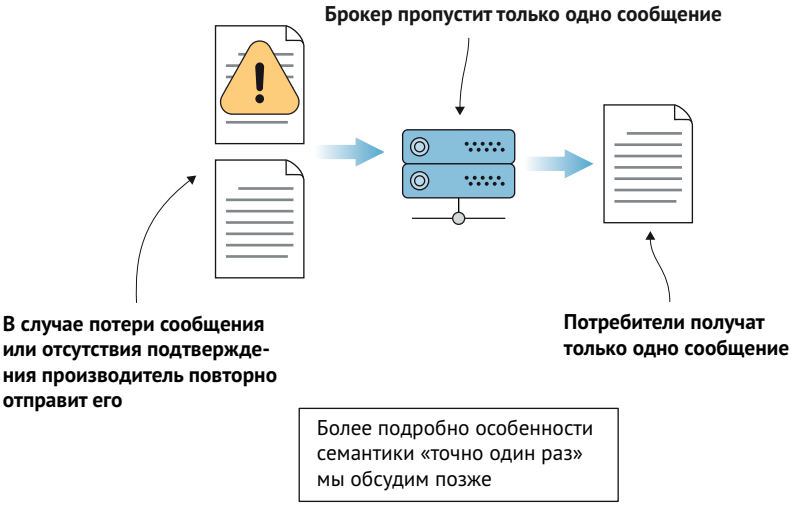

Рис. 1.5. Поток сообщений при использовании семантики «точно один раз»

Также после выхода поддержки EOS развернулась еще одна дискуссия о возможности вообще семантики «точно один раз». Корнями эта дискуссия уходит глубоко в теорию информатики, и все же вам будет полезно знать, как Kafka определяет данную семантику [4]. Если производитель отправит сообщение более одного раза, оно будет доставлено конечному потребителю только один раз. Поддержка семантики «точно один раз» затрагивает все уровни Kafka – производителей, темы, брокеров и потребителей, – и мы кратко рассмотрим их далее в этой книге, а пока продолжим наше обсуждение.

Помимо различных семантик доставки, есть еще одно общее преимущество использования брокера сообщений – если приложение-потребитель потерпело аварию или остановлено для технического обслуживания, то производитель может не ждать, пока его сообщение будет обработано. Когда потребители возобновят работу и вернутся в сеть, они смогут продолжить с того места, на котором остановились, и обработать ожидающие сообщения.

1.2. Использование Kafka

Многие традиционные компании сталкиваются со сложностями, обусловленными все возрастающей ролью технического и программного обеспечения. По этой причине возникает вопрос: как им подготовиться к будущему? Один из возможных ответов: использовать Kafka. Kafka известна как высокопроизводительная рабочая лошадка для доставки сообщений, которая по умолчанию поддерживает репликацию и обеспечивает высокую надежность.

Kafka способна удовлетворить самые взыскательные потребности в обработке данных в промышленном окружении [5]. И это – инструмент версии 1.0, которого не существовало до 2017 года! Однако, отложим эти яркие факты и зададимся вопросом: что может дать Kafka пользователям? Попробуем ответить на него дальше.

1.2.1. Kafka – разработчикам

Что может дать Kafka разработчикам? Распространение Kafka продолжает наращивать темпы, но вопросов у разработчиков не становится меньше [6]. Необходим сдвиг в традиционном подходе к обработке данных. Обмен опытом использования или устранения болевых точек может помочь разработчикам понять, почему Kafka можно считать шагом вперед в их архитектурах данных.

Одним из аспектов, способствующих переходу разработчиков на Kafka, является возможность применения прежнего опыта для познания нового. Например, многие разработчики на Java® используют привычные им концепции Spring®, поэтому в механизм внедрения зависимостей (Dependency Injection, DI) в Spring была добавлена поддержка Kafka (https://projects.spring.io/spring-kafka) и уже претерпела два выпуска. Поддерживающие проекты, а также сама платформа Kafka имеют свою расширяющуюся экосистему инструментов.

Большинство программистов сталкивалось в своей практике с проблемами, обусловленными тесной связанностью компонентов. Например, вы хотите внести изменения в одно приложение, но обнаруживаете, что они повлияют на множество других приложений, напрямую связанных с этим. Или вы начинаете модульное тестирование и оказываетесь перед необходимостью создания большого количества фиктивных объектов. Kafka, если применять ее вдумчиво, может помочь в таких ситуациях.

Возьмем, к примеру, систему управления персоналом, которая используется для оформления отпусков. Если вы привыкли к системам с поддержкой операций создания, чтения, обновления и удаления (create, read, update, delete – CRUD), то запрос на добавление записи об отпуске, скорее всего, будет обрабатываться не только бухгалтерской системой, но также системой планиро-

вания проектов для прогнозирования выполнения этапов работ. Вы свяжете эти две системы вместе? И что вы будете делать, если бухгалтерская система рухнет? Должно ли это повлиять на работу системы планирования?

Kafka позволяет отделить друг от друга приложения, которые в старых проектах часто оказывались связанными. (Более подробно об усовершенствовании текущих моделей данных мы поговорим в главе 11.) Ее можно вставить в середину рабочего процесса [7], и она будет служить единым интерфейсом к данным вместо многочисленных API и баз данных.

Некоторые говорят, что есть более простые решения. Представим себе использование процесса извлечения, преобразования и загрузки данных в базы данных для каждого приложения. Это тоже единый интерфейс для каждого приложения, и реализовать его было бы несложно, верно? Но что, если первоначальный источник данных прекратил работать или обновляться? Как часто вы проверяете наличие обновлений, и какие задержки допускаете? Кроме того, хранимые вами копии данных тоже в какой-то момент устареют или разойдутся с источником настолько далеко, что будет трудно восстановить этот поток и получить желаемые результаты. Что должно играть роль источника истины? Kafka может помочь избежать этих проблем.

Еще одна интересная тема, знакомство с которой может повысить доверие к Kafka: насколько она сама использует свои же механизмы. Например, когда мы углубимся в обсуждение потребителей в главе 5, то увидим, что Kafka использует внутренние темы для управления смещениями потребителей. В версии 0.11 семантика «точно один раз» тоже использует внутренние темы. Возможность иметь много потребителей данных, использующих одно и то же сообщение, дает множество допустимых результатов.

Еще один вопрос разработчиков может заключаться в том, почему бы не изучить Kafka Streams, ksqlDB, Apache Spark™ Streaming или другие платформы и миновать изучение ядра Kafka? Количество приложений, внутренне использующих Kafka, действительно впечатляет. Пользоваться слоями абстракции часто удобнее (а иногда без них вообще не обойтись с таким количеством движущихся частей), и все же мы считаем, что изучение самой Kafka совершенно необходимо.

Есть разница между знанием того, что Kafka – это вариант канала для Apache Flume™, и пониманием всех параметров конфигурации. Kafka Streams может упростить примеры, представленные в этой книге, но Kafka добилась успеха задолго до того, как появилась Kafka Streams. Ядро Kafka образует мощный фундамент, и его изучение, я надеюсь, поможет вам понять, почему она используется в некоторых приложениях и что происходит внутри. Если вы

хотите стать экспертом в потоковой передаче данных, то обязаны знать устройство основных распределенных частей ваших приложений и все параметры, которые можно использовать для точной настройки. С чисто технической точки зрения существует множество интересных тем информатики, применяемых на практике. Наиболее обсуждаемым, пожалуй, является понятие распределенных журналов коммитов, которое мы подробно обсудим в главе 2, и моя любимая тема – иерархические колеса синхронизации [8]. Эти примеры показывают, как Kafka решает проблему масштаба, применяя интересную структуру данных для решения практической задачи.

Мы также хотели бы отметить, что платформа распространяется с открытым исходным кодом, а это позволяет исследовать ее исходный код в поисках примеров и ответов на вопросы. Ресурсы не ограничиваются только внутренними знаниями, основанными исключительно на конкретном рабочем месте.

1.2.2. Как преподнести Kafka вашему руководству

Как это часто бывает, нередко члены высшего руководства, услышав слово *Kafka*, приходят в смущение, не задумываясь о фактическом предназначении этой платформы. Было бы неплохо объяснить им ценность этого продукта. Кроме того, нам с вами тоже полезно сделать шаг назад и посмотреть более широко на реальную ценность этого инструмента.

Одна из наиболее важных функций Kafka – возможность принимать большие объемы данных и делать их доступными для использования различными бизнес-подразделениями. Такая магистраль данных, которая делает информацию, поступающую на предприятие, доступной для всех подразделений, обеспечивает гибкость и открытость в масштабе всей компании. Потенциальным результатом является расширение доступа к данным. Большинство руководителей также знает, что, когда поступает очень много данных, возникает проблема организации максимально быстрого доступа к ним. Чтобы платить за хранение постепенно устаревающих данных на диске, их можно получать по мере их поступления и с максимальной выгодой использовать эту оперативность. Kafka дает возможность отказаться от обработки информации с помощью пакетных заданий, ограничивающих скорость превращения данных в ценность. *Быстрые данные* – более новый термин, намекающий на то, что реальная ценность кроется не только в больших объемах данных, но и в оперативности их получения.

Использование виртуальной машины Java JVM® – хорошо знакомое и привычное дело для многих центров корпоративной разработки. Возможность запуска в локальном окружении является

решающим фактором для тех, чьи данные должны управляться на месте. Неплохими вариантами также являются облачные и управляемые платформы. Платформа Kafka может масштабироваться не только вертикально, но также горизонтально, что особенно важно, учитывая существование физического предела вертикального масштабирования.

Возможно, одними из самых веских аргументов, которые можно было привести в пользу Kafka, могли бы стать яркие примеры стартапов и других компаний, действующих в той же отрасли и сумевших преодолеть когда-то непомерно высокую стоимость вычислительной мощности. Вместо крупных и мощных серверов или мейнфреймов, стоящих миллионы долларов, можно использовать менее дорогие распределенные приложения и архитектуры, которые при этом не ухудшают конкурентоспособность.

1.3. Мифы о Kafka

Приступая к изучению любой новой технологии, первым естественным желанием часто бывает сопоставить существующие знания с новыми концепциями. Этот прием тоже можно использовать при изучении Kafka, однако мы хотели бы в первую очередь отметить некоторые из наиболее распространенных заблуждений, с которыми приходилось сталкиваться в нашей работе. Мы рассмотрим их в следующих разделах.

1.3.1. Kafka работает только с Hadoop®

Как уже упоминалось, Kafka – это мощный инструмент, широко используемый в различных ситуациях. Однако многие познакомились с этой платформой в связке с экосистемой Hadoop, а кто-то – как с инструментом в составе пакета Cloudera™ или Hortonworks™. Нередко можно услышать миф о том, что Kafka работает только с Hadoop. На чем основывается такое мнение? Одна из причин заключается в том, что в Kafka применяют различные инструменты, такие как Spark Streaming и Flume, которые вместе с тем используют (или когда-то использовали) Hadoop. Еще один инструмент – Apache ZooKeeper™ – тоже часто встречается в кластерах Hadoop и может еще больше подкреплять этот миф о Kafka.

Другой часто встречающийся миф – для работы Kafka нужна распределенная файловая система Hadoop (Hadoop Distributed Filesystem, HDFS). В действительности это далеко не так. Начав знакомиться с внутренними механизмами Kafka, мы увидим, что скорость работ Kafka и обработки событий падает, когда NodeManager находится внутри процесса. Кроме того, репликация блоков, обычно являющаяся частью HDFS, выполняется иначе. Например, в Kafka реплики не восстанавливаются по умолчанию. Но даже при

том, что оба продукта используют репликацию по-разному, надежность Kafka можно легко объединить с готовностью Hadoop к сбоям по умолчанию (и, следовательно, планированием его преодоления), что является общей целью для Hadoop и Kafka.

1.3.2. Kafka ничем не отличается от других брокеров сообщений

Еще один большой миф заключается в том, что Kafka – это лишь еще один брокер сообщений. Прямые сравнения функций различных инструментов (таких как RabbitMQ™ компании Pivotal или MQSeries® компании IBM) с Kafka часто сопровождаются звездочками (или мелким шрифтом) и не всегда точно отражают варианты использования, для которых эти инструменты подходят лучше всего. Некоторые инструменты со временем приобрели или планируют приобрести новые возможности, как, например, семантика «точно один раз» в Kafka. Также можно изменить конфигурацию по умолчанию, чтобы сблизить возможности разных инструментов в том же пространстве. В целом ниже перечислены некоторые из наиболее интересных и выдающихся особенностей, которые мы вкратце рассмотрим:

- возможность повторной передачи сообщений по умолчанию;
- параллельная обработка данных.

Платформа Kafka изначально была ориентирована на работу с несколькими потребителями. Это означает, что приложение, читающее сообщение из брокера сообщений, не делает это сообщение недоступным для других приложений, которые также могут захотеть его получить и использовать. Как следствие, потребитель, уже прочитавший сообщение, сможет снова прочитать его (и другие сообщения). В некоторых архитектурных моделях, таких как лямбда (обсуждается в главе 8), ошибки программиста считаются таким же ожидаемым явлением, как и аппаратные сбои. Представьте, что вы потребляете миллионы сообщений и забываете использовать определенное поле из исходного сообщения. В некоторых очередях прочитанные сообщения удаляются или сохраняются в другом месте. Однако Kafka дает потребителям возможность отыскать конкретный момент и прочитать сообщение снова (пусть и с некоторыми ограничениями), просто просматривая более ранние позиции в теме.

Как уже было сказано, Kafka позволяет обрабатывать данные параллельно и может обслуживать нескольких потребителей в одной и той же теме. Но в Kafka также есть понятие группы потребителей, которое подробно рассматривается в главе 5. Объединяя потребителей в группу, можно определить, какие из них будут получать те или иные сообщения и как будет обслуживаться

эта группа. Группы потребителей действуют независимо друг от друга и позволяют запустить столько приложений, потребляющих сообщения в своем собственном темпе, сколько потребуется для своевременного обслуживания потока. Обработка может происходить разными способами: потреблением многими потребителями, работающими в одном приложении, и потреблением многими потребителями, работающими в нескольких приложениях. А теперь оставим в стороне другие брокеры сообщений и сосредоточимся на испытанных вариантах использования, сделавших Kafka одним из инструментов, к которому обращаются многие разработчики.

1.4. Kafka в реальном мире

Практическое использование Kafka – вот главная цель этой книги. Одна из особенностей Kafka, которую следует отметить, – про эту платформу нельзя сказать, что она решает какую-то одну конкретную задачу; она прекрасно справляется с множеством задач. Да, она у нее есть некоторые базовые идеи, и их желательно осветить в первую очередь, но не менее полезно обсудить в общих чертах некоторые реальные случаи использования Kafka. На сайте Apache Kafka перечислены основные сферы использования Kafka в реальном мире, и мы обязательно исследуем их в этой книге [9].

1.4.1. Ранние примеры

Некоторые пользователи (как и я сам) начинали с применения Kafka в роли инструмента обмена сообщениями. Лично мне после многих лет использования других инструментов, таких как IBM® WebSphere® MQ (ранее MQ Series), Kafka (в то время это была версия 0.8.3) казалась простым и удобным средством передачи сообщений из пункта A в пункт B. Kafka воздерживается от использования популярных протоколов и стандартов, таких как расширяемый протокол обмена сообщениями о присутствии (Extensible Messaging and Presence Protocol, XMPP), Java Message Service (JMS) API (ныне часть Jakarta EE) или расширенный протокол организации очереди сообщений (Advanced Message Queuing Protocol, AMQP) компании OASIS®, отдавая предпочтение нестандартному двоичному протоколу на основе TCP. Далее в книге мы рассмотрим некоторые варианты его использования.

Для конечного пользователя, разрабатывающего клиента Kafka, основная работа заключается в том, чтобы определить конфигурацию, следуя относительно простой логике (например, «Я хочу отправить сообщение в эту тему»). Наличие надежного канала для отправки сообщений также является причиной использования Kafka.

Часто хранение данных в оперативной памяти не гарантирует сохранности информации; если такой сервер отключится, то сообщения исчезнут после перезагрузки. Платформа Kafka изначально создавалась с прицелом на высокую доступность и долговременное хранение. Apache Flume предоставляет вариант канала Kafka, поддержка репликации и высокая доступность которой позволяют сделать события Flume немедленно доступными для других потребителей в случае сбоя агента Flume (или сервера, на котором он работает) [10]. Kafka позволяет создавать надежные приложения и помогает преодолевать ожидаемые сбои, с которыми рано или поздно сталкиваются распределенные приложения.

Во многих ситуациях, в том числе при попытке выбрать события, записанные распределенными приложениями, может пригодиться возможность агрегирования журналов (рис. 1.6). На рисунке показано, как журнальные записи отправляются в виде сообщений в Kafka, при этом разные приложения имеют по одной логической теме для использования этой информации. Благодаря способности Kafka обрабатывать большие объемы данных сбор событий с различных серверов или источников превратился в одну из ключевых функций. В зависимости от содержимого журнала некоторые организации используют эту функцию для аудита и выявления причин сбоев. Kafka также используется в различных инструментах журналирования (или как инструмент ввода).

Рис. 1.6. Агрегирование журналов в Kafka

Как хранение всех этих журналов не вызывает нехватки ресурсов и не мешает Kafka поддерживать высокую производитель-

ность? Передача большого количества коротких сообщений иногда может привести к перегрузке системы, потому что обработка каждого метода требует времени и дополнительных ресурсов. Поэтому для отправки и записи данных Kafka использует пакетную обработку. Особенность журналов, когда данные всегда записываются в конец, тоже помогает сэкономить ресурсы. Подробнее о форматах журналов сообщений мы поговорим в главе 7.

1.4.2. Более поздние примеры

Раньше микросервисы взаимодействовали друг с другом посредством API, таких как REST, но теперь асинхронные сервисы могут обмениваться событиями с помощью Kafka [11]. Микросервисы могут использовать Kafka как интерфейс для своих взаимодействий вместо прямых вызовов API. Kafka зарекомендовала себя как надежная платформа, позволяющая разработчикам быстро получать данные. В настоящее время многие проекты используют Kafka Streams по умолчанию, но еще до выхода Streams API в 2016 году Kafka успела зарекомендовать себя как успешное решение. Streams API можно рассматривать как слой передачи информации, обертывающий производителей и потребителей. Этот уровень абстракции реализован в виде клиентской библиотеки, позволяющей интерпретировать работу с данными как с неограниченным потоком.

В выпуске Kafka 0.11 была реализована семантика «точно один раз». Мы рассмотрим, как она работает, позже, когда поближе познакомимся с основами. Однако уже сейчас можно отметить, что сквозные рабочие нагрузки, действующие через Kafka Streams API, могут воспользоваться усиленными гарантиями доставки. Библиотека Streams упрощает этот вариант использования еще больше и избавляет пользовательскую логику приложения от любых непроизводительных расходов, гарантируя обработку сообщения только один раз от начала до конца транзакции.

Ожидается, что с течением времени количество устройств для интернета вещей (рис. 1.7) будет только увеличиваться. Все эти устройства отправляют сообщения, иногда пачками, когда подключаются к Wi-Fi или сотовой сети, и другая сторона, принимающая эти сообщения, должна иметь возможность эффективно обрабатывать их. Как вы наверняка уже поняли, огромные объемы данных – одна из областей, в которых Kafka предстает во всем блеске. Как отмечалось выше, небольшие сообщения не являются проблемой для Kafka. Датчики, сенсоры, автомобили, телефоны, умные дома и т. д. будут отправлять данные, и должно быть что-то, что обработает поток данных и сделает их доступными для выполнения тех или иных действий [12].

Это лишь небольшая подборка примеров применения Kafka на практике. Как будет показано в следующих главах, Kafka может при-

меняться в самых разных практических областях. А чтобы увидеть другие возможности практического применения этой платформы, необходимо изучать основополагающие концепции.

Рис. 1.7. Интернет вещей (Internet of Things, IoT)

1.4.3. *Когда Kafka может быть неприменима*

Kafka – отличный инструмент, который с успехом можно использовать во многих разных случаях, но нужно отметить также, что в некоторых ситуациях это не лучший выбор. Давайте рассмотрим несколько вариантов, когда полезнее использовать другие инструменты или специализированный код.

Представьте, что вам нужно получить сводные данные только раз в месяц или даже раз в год. Представьте также, что вам не требуется получать данные по запросу или повторно обрабатывать их. В таких случаях может не понадобиться, чтобы Kafka работала в течение всего года (особенно если пакет данных не особенно велик и его можно обработать целиком). Как обычно, понятие «не особенно велик» может иметь разное значение для разных пользователей.

Если в вашем случае основной операцией с данными является произвольный поиск, то Kafka будет не лучшим выбором. Линейное чтение и запись – вот основная сфера, где Kafka показывает себя с лучшей стороны и обеспечивает максимально быстрое перемещение данных. Возможно, вы слышали, что в Kafka есть индексы, но имейте в виду, что в действительности это совсем не те ин-

дексы, что используются в реляционных базах данных и конструируются на основе полей и первичных ключей.

Точно так же если нужно соблюсти точный порядок следования сообщений в Kafka для всей темы, то вам придется посмотреть, насколько практична ваша рабочая нагрузка в этой ситуации. Чтобы избежать любых нарушений в порядке следования сообщений, понадобится позаботиться о том, чтобы в системе всегда выполнялся только один поток, опрашивающий производителя, и чтобы в теме был только один раздел. Существуют разные обходные решения, но если требуется обрабатывать большие объемы данных в строго определенном порядке, то потребление данных будет ограничено одним потребителем на группу за раз.

Еще один практический момент, который приходит на ум: большие сообщения. Размер сообщения по умолчанию составляет около 1 Мбайт [13]. При передаче больших сообщений вы начнете замечать увеличение нагрузки на память. Иначе говоря, уменьшение количества сообщений, умещающихся в кеше страниц, может стать проблемой. Если вы планируете рассылать огромные архивы, то я бы посоветовал поискать более подходящие способы управления такими сообщениями. Имейте в виду, что с Kafka вы, вероятно, сможете достичь конечной цели в описанных ситуациях (в принципе, это возможно), но эта платформа может оказаться не лучшим выбором.

1.5. Онлайн-ресурсы

Сообщество Kafka проделало большую работу и создало одну из лучших (на наш взгляд) подборку документации. Kafka – часть фонда Apache (включена в Apache Incubator в 2012 году), и текущая ее документация хранится на веб-сайте проекта по адресу https://kafka.apache.org.

Еще один отличный информационный ресурс – Confluent® (https://www.confluent.io/resources). Проект Confluent был основан создателями Kafka и активно влияет на направление работы. В рамках этого проекта разрабатываются корпоративные функции и осуществляется поддержка компаний с целью помочь им в разработке их потоковых платформ. Участники проекта продолжают поддерживать открытую природу исходного кода Kafka и предоставляют презентации и лекции, в которых обсуждаются производственные задачи и достижения.

Когда в следующих главах мы начнем углубляться в другие API и параметры конфигурации, эти ресурсы станут для вас полезным справочником, где вы сможете почерпнуть дополнительные подробности. В главе 2 мы еще вернемся к теме использования Kafka и начнем знакомиться с практической стороной Apache Kafka.

Итоги

- Apache Kafka – это платформа потоковой передачи, которую можно использовать для быстрой обработки большого количества событий.
- Kafka может использоваться только в качестве шины сообщений, но в этом случае останутся неиспользованными возможности обработки данных в реальном времени.
- Для многих Kafka ассоциировалась в прошлом с другими решениями для работы с большими данными, однако в действительности Kafka является самостоятельной, масштабируемой и надежной системой. В ней используются те же системные методы обеспечения отказоустойчивости и распределенной обработки, поэтому она способна удовлетворить самые разные потребности современной инфраструктуры данных, предоставляя свои средства кластеризации.
- При использовании для потоковой передачи большого количества событий, таких как данные интернета вещей, Kafka позволяет быстро обрабатывать данные. Когда появляется дополнительная информация для ваших приложений, Kafka быстро предоставляет результаты для ваших данных, которые когда-то обрабатывались в пакетном режиме.

Ссылки

1. R. Moffatt. «The Changing Face of ETL». Confluent blog (September 17, 2018). https://www.confluent.io/blog/changing-face-etl/ (дата публикации: 10 мая 2019).
2. «Introduction». Apache Software Foundation (n.d.). https://kafka.apache.org/intro (доступно по состоянию на 30 мая 2019).
3. Документация. Apache Software Foundation (n.d.). https://kafka.apache.org/documentation/#semantics (доступно по состоянию на 30 мая 2020).
4. N. Narkhede. «Exactly-once Semantics Are Possible: Here's How Apache Kafka Does It». Блог Confluent (30 июня 2017). https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it (доступно по состоянию на 27 декабря 2017).
5. N. Narkhede. «Apache Kafka Hits 1.1 Trillion Messages Per Day – Joins the 4 Comma Club». Блог Confluent (1 сентября 2015). https://www.confluent.io/blog/apache-kafka-hits-1-1-trillion-messages-per-day-joins-the-4-comma-club/ (доступно по состоянию на 20 октября 2019).

6 L. Dauber. «The 2017 Apache Kafka Survey: Streaming Data on the Rise». Блог Confluent (4 мая 2017). https://www.confluent.io/blog/2017-apache-kafka-survey-streaming-data-on-the-rise/ (доступно по состоянию на 23 декабря 2017).

7 K. Waehner. «How to Build and Deploy Scalable Machine Learning in Production with Apache Kafka». Блог Confluent (29 сентября 2017) https://www.confluent.io/blog/build-deploy-scalable-machine-learning-production-apache-kafka/ (доступно по состоянию на 11 декабря 2018).

8 Y. Matsuda. «Apache Kafka, Purgatory, and Hierarchical Timing Wheels». Блог Confluent (28 октября 2015). https://www.confluent.io/blog/apache-kafka-purgatory-hierarchical-timing-wheels (доступно по состоянию на 20 декабря 2018).

9 «Use cases». Apache Software Foundation (n.d.). https://kafka.apache.org/uses (доступно по состоянию на 30 мая 2017).

10 «Flume 1.9.0 User Guide». Apache Software Foundation (n.d.). https://flume.apache.org/FlumeUserGuide.html (доступно по состоянию на 27 мая 2017).

11 B. Stopford. «Building a Microservices Ecosystem with Kafka Streams and KSQL». Блог Confluent (9 ноября 2017). https://www.confluent.io/blog/building-a-microservices-ecosystem-with-kafka-streams-and-ksql/ (доступно по состоянию на 1 мая 2020).

12 «Real-Time IoT Data Solution with Confluent». Документация Confluent. (n.d.). https://www.confluent.io/use-case/internet-of-things-iot/ (доступно по состоянию на 1 мая 2020).

13 Документация. Apache Software Foundation (n.d.). https://kafka.apache.org/documentation/#brokerconfigs_message.max.bytes (доступно по состоянию на 30 мая 2020).