1 Введение в Apache Kafka

В этой главе

- ✓ Общее описание и варианты применения Apache Kafka.
- ✓ Место Kafka в корпоративной экосистеме.
- ✓ Архитектура Kafka.
- ✓ Запуск и использование Kafka.

Современные корпоративные приложения часто состоят из независимых компонентов и сервисов, взаимодействующих путем обмена сообщениями. Чем больше серверов, дата-центров, облачных платформ и географических регионов включает в себя система, тем сложнее управлять потоками сообщений. Дополнительные сложности возникают из-за требований к надежности, отказоустойчивости и работе почти в реальном времени. Apache Kafka — высокопроизводительная распределенная платформа обмена сообщениями, способная эффективно обрабатывать огромные потоки данных в системах с высокими требованиями. Apache Kafka — это также распределенный программный брокер сообщений с открытым исходным кодом, разрабатываемый в рамках фонда Apache на языках Java и Scala.

В этой главе мы поговорим о базовых компонентах Apache Kafka и их назначении, а также рассмотрим возможные сферы применения и, что немаловажно, место Kafka во всех экосистемах предприятия. Наконец, мы обсудим требования к использованию Kafka: аппаратное обеспечение, инструменты и языки программирования.

1.1. Что такое Apache Kafka и как он решает наши проблемы

Мы живем здесь и сейчас. Заказывая товары на маркетплейсе, мы ожидаем, что заказ будет подтвержден сразу же, а доставку можно будет отслеживать в реальном времени. Если с картой проводится подозрительная операция, то мы хотим узнать об этом сейчас, а не завтра. Как потребители мы привыкли к тому, что организации обрабатывают данные и реагируют на них мгновенно.

Эти ожидания в сочетании с бурным ростом объемов данных — начиная от потоков информации, поступающей с датчиков Интернета вещей, и заканчивая взаимодействием потребителей с компанией по цифровым каналам — серьезно усложняют работу ИТ-команд. Традиционные архитектуры, ориентированные на пакетную обработку данных в конце дня, не вписываются в новую реальность, где счет идет на секунды.

Крупным организациям с давно не модернизируемыми системами особенно важно перейти на работу с потоками данных в реальном времени, сохраняя надежность системы. При разделении монолитных приложений на распределенные сервисы сложность системы резко возрастает. Команды развивают сервисы независимо друг от друга, но при этом должны поддерживать надежный поток данных по всей организации. Как сохранить стабильность системы при взаимодействии десятков сервисов? Как гарантировать, что обновления не нарушат работу других компонентов? И как обрабатывать огромные потоки данных, циркулирующих между системами, не создавая при этом запутанную сеть из точечных интеграций?

Для решения этих проблем Kafka предлагает архитектурный шаблон, выступая в роли центральной нервной системы для данных. Не случайно, по данным сообщества Kafka, более 80 % компаний из списка Fortune 100 используют Kafka для этих целей. Он стал основой современных распределенных архитектур, помогая обеспечивать независимое развитие сервисов и предотвращать системные сбои с помощью асинхронной коммуникации и перехода с пакетной обработки на потоковую.

Kafka — это надежная платформа с открытым исходным кодом для распределенной потоковой обработки, кардинально меняющая подход к управлению потоками данных. По сути, Kafka выступает как постоянный распределенный лог для всех событий в организации. Представьте центральную магистраль, которая надежно хранит и в реальном времени обрабатывает всю значимую информацию — от действий клиентов до изменений состояния систем. Такой подход дает несколько важных преимуществ.

- Сохранение потоков данных позволяет не только обрабатывать их в реальном времени, но и воспроизводить. Восстановив работу после сбоя, система-потребитель может начать с того места, на котором остановилась. Новый

сервис, требующий исторических данных, обработает прошлые события так же легко, как текущие.

- Благодаря своей распределенной архитектуре Kafka горизонтально масштабируется для работы с огромными объемами данных, не снижая отказоустойчивость. Один кластер Kafka обрабатывает миллионы событий в секунду, гарантируя, что данные не будут потеряны даже в случае сбоя отдельных частей системы.

Благодаря этим возможностям Kafka стал незаменимым инструментом во многих отраслях. Финансовые учреждения используют его для обработки миллионов транзакций в реальном времени и синхронизации всех систем — от обнаружения мошенничества до уведомлений клиентов. Производители направляют через Kafka потоки данных с датчиков на тысячах устройств Интернета вещей, чтобы организовывать прогнозное обслуживание и мониторинг в реальном времени. В розничной торговле Kafka применяется для управления запасами и обработки заказов.

Kafka реализует эти возможности через модель «публикация — подписка» (publisher-subscriber; pub/sub; издатель — подписчик) или, как принято ее называть в сообществе Kafka, модель «продюсер — потребитель». Продюсеры отправляют сообщения в определенные топики, а потребители обрабатывают их по мере необходимости. Но от традиционных систем обмена сообщениями Kafka отличается возможностью постоянного хранения данных (персистентностью) — данные, записанные в Kafka, можно считывать многократно в течение нескольких часов, дней или даже месяцев.

Благодаря персистентности, распределенной архитектуре и дополнительным инструментам (Kafka Connect, Kafka Streams) Kafka из высокопроизводительного брокера сообщений для LinkedIn превратился в нечто гораздо более мощное: центральную нервную систему для корпоративных данных.

Эта книга предназначена для ИТ-специалистов, которые хотят получить более глубокое представление о Kafka и способах его интеграции в существующие инфраструктуры. Она подойдет архитекторам, системным администраторам, разработчикам и инженерам данных.

Это не пособие разработчика с примерами на Java. Здесь вы найдете практические рекомендации и советы по эффективному применению Kafka в своих проектах. Наша цель — дать вам знания и навыки по использованию Kafka для потоковой передачи данных и их обработки в реальном времени в различных ИТ-средах.

1.2. Kafka в корпоративных экосистемах

Что мы имеем в виду, называя Kafka «центральной нервной системой для данных»? Идея заключается в том, что каждое *событие* (рис. 1.1) — значимое происшествие или инцидент, генерирующий данные и несущий важную информацию, — сохраняется в Kafka. В контексте корпоративных систем событие

представляет действия, изменения или транзакции в разных частях организации: взаимодействия с пользователем, обновления систем, финансовые операции и любые другие бизнес-процессы.

Рис. 1.1. Kafka выступает в роли центральной нервной системы для данных в компании. Каждое событие, происходящее на предприятии, сохраняется в Kafka. Другие сервисы могут реагировать на эти события асинхронно и обрабатывать их по своему усмотрению

В событийно-ориентированной архитектуре и в Kafka событием называется структурированный набор данных, содержащий подробную информацию о конкретном происшествии. Такие события обычно генерируются различными компонентами или приложениями предприятия, а затем публикуются в *топик* Kafka — структурную единицу организации потоков данных. Другие системы или приложения могут подписываться на эти топики, чтобы получать и обрабатывать события. При таком подходе различные части корпоративной экосистемы обмениваются данными в реальном времени.

Во многих компаниях растет разрыв между *традиционными* (legacy) системами, на которых работают текущие бизнес-процессы и модели, и новыми системами, построенными по современным принципам. В *новых системах* инновационные сервисы и решения разрабатываются с учетом современных методологий. Хотя интегрировать старые и новые системы можно и без Kafka (используя адаптеры или антикоррупционные слои[1]), он играет ключевую роль в их взаимодей-

[1] Антикоррупционный слой (anti-corruption layer, ACL) — это архитектурный шаблон проектирования в предметно-ориентированном проектировании (DDD), используемый для интеграции двух несовместимых систем, например новой микросервисной архитектуры и старой (legacy) системы. Он выступает в роли посредника, преобразующего данные и интерфейсы, чтобы внешняя, часто «грязная» модель данных не нарушала целостность новой, «чистой» доменной модели. — *Примеч. науч. ред.*

ствии. Kafka обеспечивает бесшовный обмен сообщениями между новейшими динамичными сервисами и традиционными системами, не требуя значительных изменений последних.

Традиционные системы часто не отвечают требованиям современных внутренних и внешних клиентов. Они обрабатывают данные большими пакетами с заданными интервалами, тогда как в современном динамичном мире мгновенный доступ к информации стал необходимостью. Например, актуальный остаток на банковском счете должен отображаться сразу — не через день и уж тем более не через неделю. Мы хотим в реальном времени отслеживать посылки и моментально узнавать об изменении их статуса. Современные автомобили, к примеру, генерируют огромные объемы данных, которые передаются на анализ в корпоративные центры и часто используются для работы над беспилотными автомобилями. Kafka помогает компаниям перейти от пакетной обработки в более традиционных системах к работе с потоками данных в реальном времени, позволяя мгновенно обновлять информацию, чтобы соответствовать ожиданиям клиентов и отраслевым трендам. Хотя изначально Kafka разрабатывался для обработки в реальном времени, в определенных случаях его можно использовать и для обработки пакетов, скажем, раз в день.

Подходы к разработке программного обеспечения тоже меняются. Мы больше не пытаемся вместить весь функционал в монолитные приложения, соединенные друг с другом, а разделяем большие сервисы на *микросервисы*. В этой архитектуре приложения состоят из небольших, независимо развертываемых сервисов, взаимодействующих с помощью четко определенных API. Команды разных сервисов меньше зависят друг от друга, приложения проще масштабировать, а разработка и развертывания программного обеспечения становятся более гибкими. Преимущества микросервисов полноценно реализуются при асинхронном обмене данными. Даже если один микросервис находится на обслуживании, остальные продолжают работу. Кроме того, микросервисам требуются методы коммуникации, позволяющие форматам данных в одном сервисе развиваться независимо от других сервисов. Kafka играет ключевую роль в этом контексте, предоставляя надежную платформу для асинхронной коммуникации и потоковой передачи данных и позволяя микросервисам функционировать независимо друг от друга, без помех обмениваясь информацией в слабосвязанном режиме.

Еще один тренд, обусловленный виртуализацией и распространением облачных архитектур, — возможность обходиться без специализированного оборудования. В отличие от других систем обмена сообщениями Kafka не требует выделенных устройств. Он работает на стандартном оборудовании, без специальных функций отказоусточивости, и корректно обрабатывает сбои подсистем, так что, даже если в центре обработки данных возникли проблемы, сообщения будут доставляться стабильно.

Как же Kafka обеспечивает такую надежность и производительность? Как применять его в различных сценариях и что при этом учитывать? Ответы на эти и многие другие вопросы вы найдете в этой книге.

1.3. Архитектура Kafka

Архитектура Kafka — не просто базовый фреймворк, а тщательно продуманная система для бесперебойной передачи, хранения и обработки данных. Рассмотрим ключевые компоненты Kafka (рис. 1.2), чтобы понять, почему он стал главным инструментом для распределенных приложений с потоковой обработкой данных.

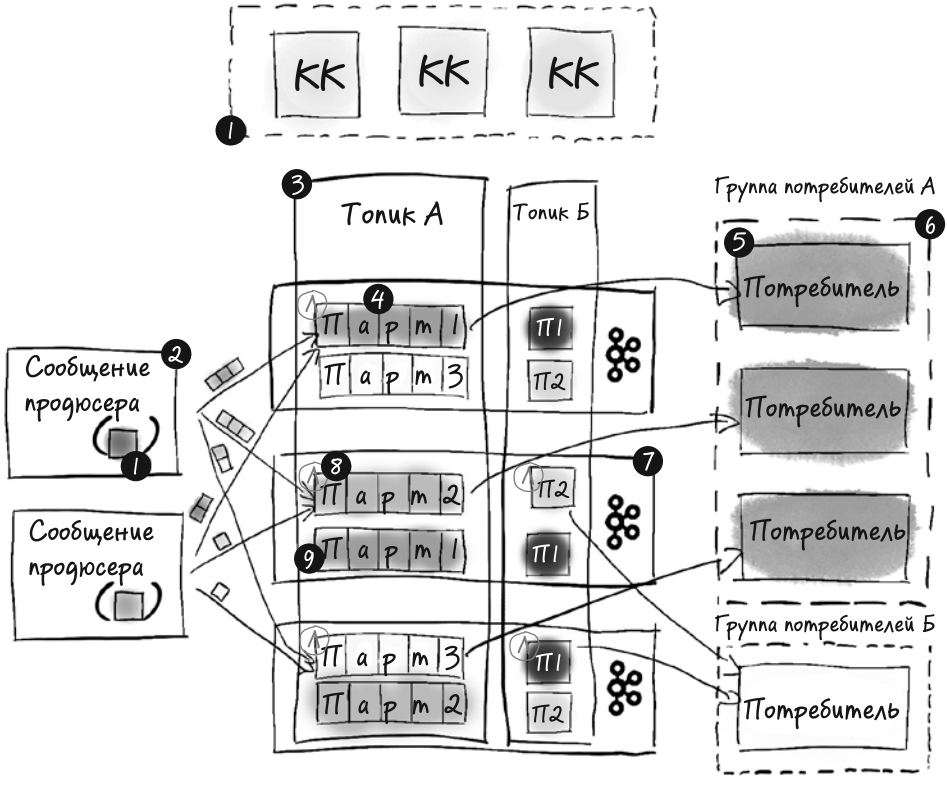

Рис. 1.2. Компоненты и поток данных в Kafka

Разработчикам и просто любопытствующим будет интересно увидеть, как организованы компоненты архитектуры Kafka и как этот инструмент определяет будущее потоковой передачи данных.

1. *Сообщения* (message) несут полезную нагрузку, передаются в виде массивов байтов и «за кадром» перед отправкой обычно группируются в пакеты.
2. *Продюсеры* (producer) отправляют сообщения лидеру партиции и выбирают партицию сами с помощью партиционера.
3. *Топики* (topic) объединяют сообщения по бизнес-темам аналогично таблицам в базе данных.

4. *Партиции* (partition) лежат в основе производительности Kafka. Деление топиков на партиции обеспечивает параллелизацию и масштабирование процессов, а партиции реплицируются на нескольких брокерах в целях отказоустойчивости и высокой доступности.
5. *Потребители* (consumer) получают и обрабатывают сообщения от Kafka, считывая данные из разных партиций и топиков.
6. *Группы потребителей* обрабатывают сообщения параллельно, позволяя масштабировать их потребление, поскольку партиции и сообщения распределяются между потребителями. При сбое одного потребителя остальные в группе берут на себя его задачи, гарантируя отказоустойчивость.
7. *Брокеры* (broker) — это серверы Kafka. Они равномерно распределяют реплики и задачи между собой, чтобы повысить производительность, и обеспечивают надежность: при отказе одного брокера его функции перехватывает другой.
8. *Лидеры* (leader) — это брокеры, ответственные за операции чтения и записи в партиции. Распределяются между всеми брокерами максимально равномерно.
9. *Фолловеры* (follower) — это брокеры, на которые копируются партиции с лидера для повышения отказоустойчивости.
10. *Координационный кластер* (Kafka Raft (KRaft), ранее — кворум, или кластер, ZooKeeper) используется Kafka для координации собственной работы.

Рассмотрим взаимодействие компонентов Kafka на примере. Представьте банковское приложение, обрабатывающее денежные переводы. Это продюсер, который генерирует сообщения о каждом событии перевода. В сообщении указаны счет отправителя, счет получателя, сумма и время операции.

Это сообщение не будет направлено потребителю напрямую — сначала оно попадет в топик Kafka с именем `bank-transfers`. Топик здесь можно сравнить с категорией, по которой объединяются сообщения.

Этот топик состоит из нескольких партиций, и каждая из них обрабатывает часть поступающих сообщений, таким образом обеспечивая параллельную обработку. В нашем случае партиции могут управлять денежными переводами разных типов.

Кластер Kafka, состоящий из брокеров, обеспечивает функционирование всей системы. Брокеры — это серверы, которые хранят данные и управляют ими. Вместе они составляют отказоустойчивый и масштабируемый кластер Kafka. Каждый брокер управляет одной или несколькими партициями каждого топика, обеспечивая эффективное распределение данных.

В традиционных конфигурациях Kafka для координации брокеров и управления метаданными использовался ZooKeeper, который играл ключевую роль

в поддержании распределенной работы Kafka. В более новых версиях Kafka обходится без ZooKeeper — теперь для внутренней координации применяется протокол KRaft (упоминавшийся выше), так что Kafka управляет координацией напрямую, без внешней системы.

> **ПРИМЕЧАНИЕ** Подробности перехода с ZooKeeper на KRaft будут описаны в следующих главах. Пока достаточно знать, что он был нужен для повышения масштабируемости и упрощения архитектуры.

Теперь рассмотрим происходящее с точки зрения потребителей. Это приложения или сервисы, которые читают сообщения из топиков Kafka. Для эффективного чтения потребители объединяются в группы. Каждая группа состоит из нескольких потребителей, а каждая партиция доступна для чтения только одному потребителю в группе. Параллельная обработка сообщений выполняется быстрее.

У каждой партиции есть лидер и несколько фолловеров. Лидер отвечает за операции чтения и записи, а фолловеры реплицируют данные, чтобы обеспечить отказоустойчивость. При отказе лидера его заменяет фолловер, так что поток данных не прерывается.

Итак, сообщения о банковских переводах в Kafka эффективно обрабатываются группами потребителей, благодаря чему на счете сразу отображается корректный остаток. Кластер Kafka с его распределенной архитектурой и механизмами отказоустойчивости формирует основу для этого надежного и масштабируемого потока данных. Kafka упорядочивает потоки событий, управляя лидерами и фолловерами, распределяя сообщения по партициям и координируя работу брокеров.

1.4. Запуск и использование Kafka

Для эффективного запуска Kafka необходимо несколько компонентов и условий. В первую очередь требуется надежный и правильно настроенный набор серверов для размещения кластера Kafka. Каждый сервер в кластере выступает в роли брокера и участвует в распределенной обработке потоков данных. Кроме того, нужна адекватная сетевая инфраструктура с низкой задержкой и высокой пропускной способностью, которая сможет обеспечить бесперебойное взаимодействие между брокерами Kafka. Пользователи должны четко понимать требования к данным и проектировать топики и партиции так, чтобы данные распределялись и упорядочивались эффективно. Продюсеры и потребители нужно написать таким образом, чтобы они эффективно взаимодействовали с Kafka, обеспечивая отправку, прием и обработку данных в реальном времени. Наконец, требуется комплексная стратегия мониторинга и управления, позволяющая отслеживать производительность Kafka и оперативно устранять неполадки. Для успешной

работы нужны правильно настроенная инфраструктура, надежная сеть и хорошее понимание архитектуры Kafka и связанных компонентов.

Самостоятельно управлять Kafka можно, но сложно. В облаке многие вендоры предлагают Kafka в виде управляемого решения. Например, Amazon Managed Streaming for Apache Kafka (Amazon MSK) или Azure HDInsight. Кроме того, существуют специализированные вендоры, такие как Confluent и Aiven. Можно также использовать сервисы с поддержкой Kafka, наподобие Redpanda или WarpStream, которые обещают дополнительные функции или больше производительности за меньшие деньги. Эти предложения выглядят интересно, но у нас недостаточно опыта работы с ними, чтобы можно было что-то посоветовать. Нужно понимать, что инфраструктура Kafka — только часть успешной потоковой архитектуры, поэтому, даже если в компании используются управляемые сервисы, в ней должны быть свои специалисты.

> **СОВЕТ** У нас был успешный опыт работы с Confluent и Aiven, так что мы можем их рекомендовать.

Помимо упомянутых инфраструктурных компонентов, потребуются определенные языки программирования и инструменты. Сам Kafka реализован на Scala и Java, поэтому на серверах, на которых размещаются брокеры Kafka, должна быть установлена среда выполнения Java (Java Runtime Environment, JRE). Для взаимодействия с Kafka и разработки приложений-продюсеров и приложений-потребителей, как правило, применяются Java, Scala, Python и другие языки, поддерживаемые клиентами Kafka. В целях упрощения интеграции Kafka предлагает официальные клиентские библиотеки для различных языков.

1.5. Траектория обучения

В последующих главах мы последовательно рассмотрим Kafka, плавно переходя от базовых концепций к внутренним нюансам работы. В книге много схем и рисунков, наглядно иллюстрирующих ключевые принципы, архитектуру и процессы Kafka. Они помогут вам лучше понять распределенную архитектуру, а также устройство и взаимодействие топиков, партиций и брокеров.

Обучение строится на практическом подходе — мы используем простые, но эффективные примеры, позволяющие сразу приступить к работе с Kafka. Прикладные сценарии и пошаговые инструкции помогут вам научиться применять полученные знания для создания и запуска приложений Kafka. Все примеры — от создания и потребления сообщений до настройки и управления кластерами Kafka — продуманы так, чтобы вы могли экспериментировать с ними и оттачивать навыки. С помощью наглядных иллюстраций и практических примеров мы разъясним все сложности Kafka и поможем применить полученный опыт в вашей работе.

Резюме

- Kafka — это мощная распределенная платформа для потоковой обработки данных, работающая по модели «публикация — подписка» и обеспечивающая беспрепятственный поток данных между продюсерами и потребителями.
- Kafka широко применяется в различных отраслях и превосходно справляется с задачами анализа данных в реальном времени, генерации событий (event sourcing), агрегации логов и потоковой обработки, помогая организациям принимать решения на основе актуальных данных.
- Архитектура Kafka ориентирована на отказоустойчивость, масштабируемость и долговечность, гарантируя надежную передачу и хранение данных даже при системных сбоях.
- В финансах, розничной торговле, телекоммуникациях и других сферах Kafka используется для обнаружения мошенничества в реальном времени, обработки транзакций, управления запасами, обработки заказов, мониторинга сетей и обработки масштабных потоков данных.
- Помимо основной системы обмена сообщениями, в экосистему Kafka входят Kafka Connect (коннекторы для внешних систем) и Kafka Streams (библиотека для разработки потоковых приложений), повышающие общую полезность платформы.
- Kafka может служить центральным узлом для интеграции разнородных систем.
- Продюсеры отправляют сообщения в Kafka для распределения.
- Потребители получают сообщения из Kafka и обрабатывают их.
- Топики организуют сообщения по каналам, или категориям.
- Партиции разделяют топики, обеспечивая параллельную обработку и масштабирование процессов.
- Брокеры — это серверы Kafka, управляющие хранением, распределением и извлечением данных.
- KRaft или ZooKeeper координирует задачи в кластере Kafka.
- Kafka обеспечивает устойчивость данных, используя репликацию.
- Kafka масштабируется горизонтально путем добавления брокеров в кластер.
- Kafka работает на стандартном оборудовании.
- Kafka реализован на Java и Scala, но предлагает клиенты и для других языков (например, Python).