Ben Stopford

DESIGNING EVENT-DRIVEN SYSTEMS

Concepts and Patterns for Streaming Services with Apache Kafka

O'Reilly

Бен Стопфорд

ПРОЕКТИРОВАНИЕ СОБЫТИЙНО-ОРИЕНТИРОВАННЫХ СИСТЕМ

Концепции и шаблоны проектирования сервисов потоковой обработки данных с использованием Apache Kafka

Перевод с английского

Иркутск
ITSumma Press
2019

УДК 004.451
ББК 32.972.34
С81

Главный редактор	***Анастасия Овсянникова***
Научные редакторы	***Дмитрий Чумак, Иван Сидоров***
Перевод	***Владимир Жданов***
Корректоры	***Лариса Ковтун, Евгений Финкельштейн***
Верстка	***Неля Алькова***
Иллюстрация на обложке	***Иннокентий Астафьев***

Впервые на русском языке

Стопфорд, Бен

Проектирование событийно-ориентированных систем: Концепции и шаблоны проектирования сервисов потоковой обработки данных с использованием Apache Kafka / Бен Стопфорд ; Пер. с англ. — 2-е изд., испр. — Иркутск : ITSumma Press, 2019. – 175 с.

Современный мир плотно покрыт коммуникационными сетями разного уровня, и с лёгкой руки его можно назвать информационно связанным. Но, несмотря на это, всё ещё есть множество ситуаций, когда различным людям, командам и системам предпочтительнее взаимодействовать асинхронно, обмениваясь информацией лишь изредка, от случая к случаю, но при этом сохранять погружённость в общие контексты. Добиться этого в привычных императивных системах довольно сложно. Что делает подход, предлагаемый создателями Apache Kafka, особенно ценным. Данная книга поможет всем заинтересованным ознакомиться с общими принципами построения слабосвязанных событийно-ориентированных архитектур и раскроет основные преимущества и недостатки применяемых подходов.

ISBN 978-5-6042412-1-9

ISBN 978-1492038238
англ.

Оглавление

Вступительное слово от ITSumma Press vii
Предисловие от научного рецензента ix
Вступление xi
Предисловие xv

Часть I
Расставляем декорации

Глава 1. Введение 3
Глава 2. Истоки потоковой обработки 9
Глава 3. Всё ли вы знаете о Kafka? 13
Kafka это REST, только асинхронный? 13
Похожа ли Kafka на сервисную шину? 14
Похожа ли Kafka на базу данных? 15
Что такое Kafka на самом деле? Платформа потоковой обработки данных 16
Глава 4. Не только обмен сообщениями: обзор брокера Kafka 19
Журнал: эффективная структура для сохранения и распространения сообщений 20
Линейная масштабируемость 22
Разделение нагрузки в мультисервисных экосистемах 23
Выполнение гарантий строгой упорядоченности 24
Обеспечение надёжности сообщений 25
Балансируем нагрузку сервисов и делаем их высокодоступными 26
Уплотнённые темы 27
Долговременное хранение данных 28
Безопасность 28
Заключение 29

Часть II
Проектирование событийно-ориентированных систем

Глава 5. События — основа сотрудничества 33

Команды, события, запросы ... 34
Связанность и брокеры сообщений ... 36
Всегда ли хороша слабая связанность? ... 37
Связанность данных неизбежна ... 38
Использование событий для уведомлений ... 39
Использование событий для передачи состояния ... 41
Какой подход использовать ... 42
Шаблон «событийное сотрудничество» ... 44
Взаимодействие с потоковой обработкой ... 45
Смешение подходов: событийно-ориентированный и «запрос-ответ» ... 47
Заключение ... 49
Глава 6. Обработка событий с помощью функций состояния ... 51
Делаем сервисы контекстно-зависимыми ... 53
Событийно-ориентированный подход ... 54
Чистый (контекстно-независимый) потоковый подход ... 55
Контекстно-зависимый потоковый подход ... 56
Практические вопросы контекстозависимости ... 58
Заключение ... 59
Глава 7. Источник событий, CQRS и другие модели состояний ... 61
Коротко про источники событий, источники команд и CQRS ... 61
Контроль версий ваших данных ... 63
Превращение событий в источник истины ... 66
Командно-запросная изоляция ... 67
Материализованные представления ... 68
Полиглотные представления ... 70
Полный факт или изменения? ... 70
Реализация источников событий и CQRS с помощью Kafka ... 72
Создание представлений в Kafka Streams с помощью таблиц и хранилищ состояний ... 72
Пишем данные в тему Kafka из базы данных с помощью Kafka Connect ... 73
Пишем данные в тему Kafka из хранилища состояний с помощью Kafka Streams ... 74

Интеграция со старыми системами с помощью CDC 75
Делаем запросы к оптимизированному для чтения представлению в БД 76
Образы в памяти и предзаполненные кэши 77
Представление источника событий 78
Заключение 79

Часть III
Переосмысление архитектуры в масштабе компании

Глава 8. Совместное использование данных и сервисов в организации 83
Инкапсуляция не всегда идёт на пользу 86
Дихотомия данных 88
Что происходит с системами по мере их развития? 88
Проблема Главного сервиса 89
Проблема REST-ETL 89
Делайте внешние данные первоочередными 91
Не бойтесь развиваться 92
Заключение 92
Глава 9. Поток событий как общий источник истины 95
База данных наизнанку 95
Заключение 98
Глава 10. Компактные данные 101
Базам данных необязательно запоминать то, что уже хранится в системе обмена сообщениями 101
Отбросьте всё лишнее, берите только необходимые данные 103
Пересборка представлений источника событий 104
Kafka Streams 104
Базы данных и кэши 104
Решение затруднений при перемещении данных 105
Автоматизация и миграция схем 105
Проблема расхождения данных 106
Заключение 108

Часть IV
Согласованность, параллелизм, эволюция парадигм

Глава 11. Согласованность и параллелизм в событийно-ориентированных системах 111

Итоговая согласованность ... 113
Своевременность ... 115
Коллизии и объединение сообщений ... 116
Принцип единственного писателя ... 117
Управляющая тема ... 119
Единственный писатель для смены состояния ... 119
Атомарность транзакций ... 120
Идентификация и контроль параллелизма ... 120
Ограничения ... 122
Заключение ... 122
Глава 12. Новый взгляд на транзакции ... 123
Проблема дублирования ... 123
Использование транзакционных API для удаления дублей ... 126
Доставка строго один раз — это одновременно и идемпотентность, и атомарная фиксация изменений ... 127
Как работают транзакции в Kafka ... 128
Атомарная запись состояний и событий ... 130
Нужны ли нам транзакции? Можно ли добиться того же результата с помощью идемпотентности? ... 131
Чего не могут делать транзакции? ... 132
Использование транзакций в сервисах ... 133
Заключение ... 133
Глава 13. Изменение данных и схем с течением времени ... 135
Управление изменением данных с помощью схем ... 135
Управление изменениями в схеме данных и нарушение обратной совместимости ... 136
Совместная работа над изменением схемы ... 139
Обработка нечитаемых сообщений ... 140
Удаление данных ... 140
Запуск удалений в нижестоящих базах данных ... 142
Разделение публичных и частных тем ... 142
Заключение ... 142
Часть V
Создание потоковых сервисов с помощью Kafka
Глава 14. Kafka Streams и KSQL ... 147

Разработка простого сервиса отправки почты с помощью Kafka Streams и KSQL 147
Оконные функции, соединения, таблицы и хранилище состояний 150
Заключение 152
Глава 15. Построение потоковых сервисов 153
Экосистема подтверждения заказа 153
Объединение-фильтрация-обработка 154
Представление на основе событий в Kafka Streams 155
Нарушение принципа CQRS через блокировку чтения 156
Масштабирование параллельных операций в потоковых системах 156
Переназначение ключей для соединения 160
Перераспределение данных и поэтапное исполнение 161
Ожидание N событий 162
Размышления на тему архитектуры 162
Более комплексная потоковая экосистема 163
Заключение 165
Об авторе 167
Список литературы 169

Вступительное слово от ITSumma Press

Мы с волнением и радостью представляем вам первую книгу нашего издательства. Почему мы, на сто процентов технологическая компания, решили заняться таким, казалось бы, консервативным направлением, как издание книг?

Многие считают, что книги уже не в моде. Однако у нас материальные источники информации вызывают большое уважение. Особенно когда это источник такой информации, которая не требует немедленного практического применения. Когда нет нужды с руководством на одном мониторе писать код на втором мониторе, а на третьем сразу проверять полученные результаты. Когда книга — это скорее ресурс для размышлений о подходах к технологиям, для комплексного анализа и выводов, к которым можно будет обращаться за советом еще долгое время. Чего не скажешь про прикладные учебники и руководства, быстро теряющие свою актуальность в современных реалиях. Эта книга будет интересна в течение многих лет после издания необычностью предлагаемых подходов к решению набивших оскомину проблем. Именно такого рода книги мы и хотели бы представить вниманию наших читателей.

Работа Бена Стопфорда стала нашим первым выбором не случайно. В ней автор позволяет взглянуть на устоявшиеся подходы к проектированию программных (веб-ориентированных и не только) систем с довольно непривычной для современного разработчика точки зрения. Как один из главных идеологов и создателей самой популярной на данный момент системы обмена сообщениями Kafka, Стопфорд вкладывает всю душу в то, чтобы завлечь читателя в свой мир бесконечного обмена событиями, асинхронных взаимодействий и конвейерной обработки данных.

Автор подвергает сомнению желание многих продолжать внедрять устоявшиеся монолитные схемы. Базы данных — это больше не центр вселенной, мир не крутится вокруг них. Десятки команд могут выпускать различные части единой системы совершенно независимо друг от друга, и при этом стабильность самой системы никак не пострадает.

Звучит несколько безумно? Или нет?

Ну и конечно же, будем рады обсудить с нашими читателями их впечатления, выводы, предложения. Возможно даже, кто-то захочет поделиться своими необычными «вечными» книгами, которые перевернули его взгляд на мир технологий, но до сих пор не были переведены на русский язык. Напишите нам о них.

До связи!

Предисловие от научного рецензента

В своей деятельности в компании ITSumma мы встречаемся и с огромным количеством технологий как таковых, и с самыми разнообразными архитектурными подходами. В том числе и с событийно-ориентированными системами, которые неразрывно связаны с микросервисной архитектурой. Последняя стала неотъемлемой частью индустрии.

Однако, кроме самого концепта разделения монолита на микросервисы, всё ещё остаются вопросы об их взаимодействии. Зачастую микросервисная архитектура выглядит как «технология ради технологии», в результате ухудшающая качество продукта. А переход на неё — как поиск «серебряной пули». В таком случае потоки данных между сервисами становятся во главу угла.

В нынешних условиях нескончаемого потока информации, микросервисы должны не просто общаться между собой по какому-то протоколу, но и обрабатывать гигантские потоки данных в реальном времени. Одно из направлений в проектировании таких систем — событийно-ориентированное взаимодействие между микросервисами. В своей книге Бен Стопфорд детально рассмотрел подходы по построению таких архитектур. В качестве основной системы автор приводит Apache Kafka — не просто как брокер сообщений, а как многофункциональный инструмент для микросервисного взаимодействия и обработки данных.

В книге изложены подходы, которые только начали применяться, несмотря на то, что теоретическая база уже достаточно хорошо проработана. На момент подготовки русскоязычного издания для большинства определений, упоминаемых автором, еще не было устоявшегося перевода с английского. Поэтому отдельное внимание мы уделили работе с терминологией и разъяснению некоторых понятий. Кроме того, используя наш повседневный опыт работы со схожими системами, мы постарались подойти к переводу книги не как к подготовке очередного типового технического издания, а вложить весь наш накопленный технологический опыт. Создать реально полезное, понятное, не пестрящее англицизмами и не искажающее смысл издание, листая которое хотелось бы углубиться в

чтение, а не ограничиться скептическим «лучше оригинал почитаю», как это обычно бывает с переводами технической литературы.

— Иван Сидоров
Исполнительный директор компании ITSumma

Вступление

Всё то долгое время, что мы говорим о сервисах, мы говорим о данных. На самом деле, прежде чем в нашем лексиконе появился термин «микросервисы», в старые-добрые времена сервис-ориентированных архитектур, мы говорили о данных: как получать к ним доступ, где их хранить, кто ими «владеет». И хотя данные являются главным фактором успеха бизнеса в долгосрочной перспективе, довольно часто они же становятся камнем преткновения при проектировании и развитии систем.

Моё собственное путешествие в мир микросервисов началось с работы, которую я выполнял, помогая различным организациям выпускать своё программное обеспечение (ПО) быстрее. В итоге я тратил огромное количество времени на такие задачи, как периодический анализ временных затрат, построение конвейеров сборки, автоматизацию тестирования и развёртывание инфраструктуры. Появление облаков стало огромным благом, так как автоматизация развёртывания инфраструктуры позволила нам быть ещё более продуктивными. Но некоторые фундаментальные проблемы всё же остались. Зачастую приложение было разработано так, что его было очень сложно выкладывать. И данные были ядром проблемы.

В то время самым частым приёмом, который мне доводилось видеть в сервис-ориентированных системах, было разделение одной базы данных (БД) между несколькими сервисами. Причины были просты: необходимые мне данные уже лежат в другой БД, к которой у меня есть доступ; поэтому мне оставалось лишь зайти в эту БД и взять то, что мне нужно. На начальном этапе это способствует быстрой разработке нового сервиса, но с течением времени такой подход становится источником больших проблем.

Как я подробно описывал в моей книге «Построение микросервисов»,[1] разделяемая БД создаёт в вашей архитектуре значительный узел зацепления. Становится сложно понимать, какие изменения можно вносить в структуру БД, разделяемой между множеством отдельных сервисов.

Дэвид Парнас[2] показал нам в 1971 году, что секрет разработки программного обеспечения, составные части которого могут меняться

независимо — это скрывать информацию этих частей друг от друга. Ведь, условно, разделение БД между различными сервисами ограничивает возможность независимого развития наших кодовых баз.

Вместе с изменением требований и ожиданий от ПО изменилась и организация ИТ-команд. Переход от разрозненных ИТ-департаментов к бизнес- или продукто-ориентированным командам помог улучшить клиенто-ориентированность этих команд. Этот переход часто происходил параллельно с желанием повысить автономность этих команд, позволить им развивать новые идеи и внедрять их, вместе с тем снижая необходимость координации с другими частями организации. Но сильно связанные архитектуры требуют значительной координации между системами и поддерживающими их командами и, следовательно, являются врагом любой организации, желающей повысить автономность.

В Amazon это поняли много лет назад. Они хотели повысить автономность команд разработчиков, чтобы позволить компании развиваться и разрабатывать ПО быстрее. И с этой целью были созданы маленькие, независимые команды, которые полностью контролировали весь цикл разработки ПО. Стив Йегге после своего ухода из Amazon в Google пытался ухватить идею того, что именно помогало группам разработчиков работать так хорошо в его печально известной (в определённых кругах) «Тираде о платформах (англ. Platforms Rant)».[3] В ней он выделил наказ основателя Amazon Джеффа Безоса относительно того, как команды должны взаимодействовать, и как они должны проектировать системы. И, в частности, эти пункты перекликаются с моими мыслями:

> 1) отныне все команды будут предоставлять свои данные и функции через сервисный интерфейс;
> 2) команды должны взаимодействовать друг с другом через эти интерфейсы;
> 3) никакой другой формы межпроцессного взаимодействия разрешено не будет: ни прямых ссылок, ни непосредственных чтений из баз данных другой команды, ни моделей разделяемой памяти, никаких обходных трюков. Единственным допустимым способом коммуникации остаются сетевые запросы через сервисный интерфейс.

Я же пришёл к осознанию того, что способ хранения и передачи данных имеет ключевое значение для разработки слабосвязанных архитектур.

Первостепенную важность имеют чётко определённые интерфейсы и сокрытие информации. Если нам надо хранить данные в базе, эта база должна быть частью сервиса, недоступной напрямую для других сервисов. Чётко определённый интерфейс должен описывать, когда и как данные можно получать и обрабатывать.

Большую часть последних нескольких лет я потратил на продвижение

этой идеи. Но, несмотря на то, что всё больше людей принимают эту идею, сложности остаются. В реальном мире сервисам нужно работать совместно и иногда необходимо делиться данными. Как сделать это эффективно? Как убедиться, что обмен данными выстроен с учетом требований к нагрузкам и сетевым задержкам вашего приложения? Что происходит, когда одному сервису требуется большое количество информации от другого?

Начните использовать потоки событий, а конкретнее — потоки вроде тех, что организовывают Kafka и похожие технологии. Мы уже используем брокеры для обмена событиями, но способность Kafka делать этот поток устойчивым позволяет нам рассмотреть новый способ хранения и обмена данных без потери возможности создания слабосвязанных автономных архитектур. В этой книге Бен говорит об идее «выворачивания базы данных наизнанку» — понятии, которое, я подозреваю, получит столько же скептических отзывов, сколько получил в своё время я, когда продвигал идею ухода от гигантских разделяемых БД. Но за последнюю пару лет, что я провёл, исследуя эти идеи с Беном, я не могу отделаться от мысли, что и он, и другие люди, работающие над этими понятиями и технологиями (конечно, с огромным количеством прототипов), действительно, что-то поняли.

Я надеюсь, что идеи, изложенные в этой книге, станут следующей ступенью нашего представления о совместном использовании и обмене данными, помогая нам изменить подход в построении микросервисных архитектур. Эти идеи могут показаться странными на первый взгляд, но к ним стоит присмотреться. Бен устроит вам очень интересное путешествие.

— Сэм Ньюман

Предисловие

В 2006 году я работал в компании ThoughtWorks в Великобритании. В то время наш офис был полон энергии, мы занимались кучей интересных вещей. Движение Agile было на пике, разработка через поведение (англ. behavior-driven development) процветала, люди экспериментировали с источниками событий (англ. event sourcing), а сервис-ориентированные архитектуры применялись в более мелких проектах, чтобы в итоге помочь разобраться с некоторыми из проблем, которые мы встречали в крупных проектах.

Одним из проектов, над которым я работал, руководил Дэйв Фарли, энергичный и весёлый парень, который смог привнести свою энергию практически во всё, что мы делали. Проект был относительно стандартным, средних размеров корпоративным приложением. Это был веб-портал, где клиенты могли запрашивать различные юридические услуги, касающиеся прав собственности на недвижимость. В ответ система запускала различные синхронные и асинхронные процессы, чтобы привести в действие множество сервисов.

В этом конкретном проекте был ряд интересных элементов, но единственной деталью, которая вводила меня в ступор, было то, как сервисы взаимодействовали между собой. Это была первая система, над которой я работал, построенная исключительно на событийном взаимодействии. Ранее я участвовал в разработке нескольких сервис-ориентированных систем, построенных на удалённых вызовах процедур (англ. remote procedure calls, далее — RPC) или системах обмена «запрос-ответ», и новый проект мне показался совершенно другим. Было что-то волшебное в том, что вы могли просто подключать новые сервисы прямо к потоку событий и с чувством глубочайшего удовлетворения наблюдать за тем, как система со свистом обрабатывает данные.

Несколько лет спустя я работал в крупной финансовой организации, которая хотела построить сервис данных в самом сердце компании, где приложения могли бы найти наборы данных необходимые для работы банка — сделки, оценки, справочные данные и тому подобное. Я считаю такого рода задачу достаточно привлекательной: она была технически сложной, и, хотя ряд банков и других крупных компаний использовали

такой подход и раньше, мне казалось, что технологии продвинулись к этапу, когда мы могли бы построить что-то действительно интересное и инновационное.

Но правильное понимание технологии было только началом решения проблемы. Система должна была взаимодействовать с каждым крупным отделом, что подразумевало огромное количество заинтересованных сторон с большим потоком требований, многочисленные расписания выпуска новых версий и разноплановые ожидания по продолжительности безотказной работы. Я помню, как мы обсуждали практические детали проекта, проговаривая его структуру во время двухнедельного установочного совещания. И хотя задача казалась довольно трудной не только технически, но и организационно, мы верили, что сможем ее выполнить.

В итоге мы собрали команду с кучей людей из ThoughtWorks, Google и нескольких других мест и создали систему с довольно интересными свойствами. Вычислительный кластер держал все запрашиваемые данные в памяти, распределённой среди более чем 35 машин в каждом дата-центре. Это позволяло выдержать запросы вычислительной сети. Записи шли напрямую через слой запроса в систему обмена сообщениями, которая создавала (что было несколько необычно для того времени) систему записи (англ. system of record). Как слой запроса, так и слой обмена сообщениями были спроектированы для дальнейшей сегментации, которая позволяла им линейно масштабироваться. Поэтому каждый запрос на вставку или изменение публиковался еще и как событие, без вариантов, ведь это было заложено в сердце архитектуры.

Интересно, что при превращении системы обмена сообщениями в систему записи поток данных можно преобразовать во множество полезных вещей: записывать его резервную копию на диск, перенаправлять его в другой дата-центр, формировать различные аналитические базы данных для построения отчетов, и, конечно, предоставлять доступ к потоку данных всем желающим, через какое-либо API (программный интерфейс приложения, англ. application programming interface).

Но в то время я еще не осознавал реальную важность использования системы обмена сообщениями в качестве системы записи. Я помню, как после моей презентации о проекте на конференции QCon один слайд «Обмен сообщениями как система записи», который я едва не перелистнул, зачитав лишь название, вызвал больше вопросов, чем весь материал о гармонично распределённом слое слияния, о котором я не замолкал всю презентацию. Так постепенно стало ясно, что большинству клиентов по-настоящему достаточно чего-то более простого, чем все эти штуки, типа предварительного кэширования, ориентированного на данные

и ускоряющего операции соединения (англ. join); интерфейса по типу SQL-через-Документ (англ. SQL-over-Document interface); неизменяемых моделей данных; и схем отложенного связывания. В то время как они начинали использовать сервис данных по прямому назначению, по прошествии времени нужда заставляла их делать копии, хранить их независимо и, в итоге, обрабатывать самостоятельно. Но, несмотря на это, они все-таки считали центральный набор данных целесообразным и часто создавали дополнительный поднабор, а затем возвращались для его расширения. Поэтому, поразмыслив, казалось, что система обмена сообщениями, оптимизированная для хранения наборов данных, будет более уместна, чем база данных, оптимизированная для их создания. Немного позже появился Confluent, и Kafka стала идеальным решением для этого типа проблем.

Интересно, что два этих проекта (приложение для купли-продажи и сервис общебанковских данных) имели больше общего, чем изначально могло показаться. Приложение для купли-продажи отлично поддерживало совместную работу и при этом его было легко встроить. В работе с банком, гораздо больший набор приложений и сервисов интегрировался между собой посредством событий, имея при этом возможность делать выборки по истории событий. И, несмотря на то, что предметные области проектов достаточно сильно отличались (небольшое приложение и ПО для целой компании), их общей особенностью был механизм использования событий.

Сегодня, потоковые системы во многом отличаются от вышеприведённых примеров, но базовые характеристики остались неизменными. Тем не менее, дьявол кроется в деталях, и за последние несколько лет мы видели, как клиенты использовали разнообразные подходы к решению подобного рода задач. В этой короткой книге я попытался извлечь ключевые элементы этих подходов, в которых одинаково активно применяются как системы распределённого журналирования (англ. distributed logs), так и инструменты потоковой обработки.

Как читать эту книгу

Книга состоит из пяти частей. Часть I базовая. В ней происходит знакомство с Kafka и потоковой обработкой данных и проводится полезный даже для опытных читателей обзор базовых понятий. В Части II вы узнаете, как строить событийно-ориентированные системы, как такие системы связаны с потоковой обработкой с состоянием (англ. stateful stream processing), и как применить такие шаблоны событийное сотрудничество, источник событий, и командно-запросная изоляция (англ. command query responsibility segregation, далее — CQRS). Часть

III носит больше теоретический характер, в ней мы возьмём основные идеи из Части II и приложим их к структуре целой организации. Здесь мы ставим под сомнение многие из распространённых ныне подходов и углубляемся в такие шаблоны, как Потоки событий как источник истины. Части IV и V практические. В Части V начинается первое погружение в программирование. Мы создали связанный проект на GitHub, который станет для вас стартовой площадкой, если вы захотите начать строить небольшие сервисы, используя Kafka Streams.

Вступление, данное в Главе 1, представляет поверхностный обзор основных понятий, затронутых в этой книге, так что это хорошее место для старта.

Благодарности

Многие люди внесли свой вклад в эту книгу, как прямо, так и косвенно, но особую благодарность хотелось бы выразить следующим людям: Джей Крепс, Сэм Ньюман, Эдвард Рибейро, Гвен Шапира, Стив Каунсел, Мартин Клепман, Ева Бизек, Ден Хэнли, Тим Бергланд, и, конечно же, моя всетерпящая жена Эмили.