Оглавление

Об авторе

Сэм Ньюмен — разработчик, архитектор, писатель и оратор, который работал с разными компаниями в разных областях по всему миру. Он работает независимо, концентрируя свое внимание в основном на облачных вычислениях, непрерывной доставке и микрослужбах. Его предыдущая книга — бестселлер "Создание микросервисов", также изданный в издательстве O'Reilly.

Когда он не "прыгает с одной подножки вагона на другую", его можно найти в сельской местности Восточного Кента, занимающимся различными видами спорта.

Предисловие

Еще несколько лет назад некоторые из нас лишь поговаривали о том, что, дескать, микрослужбы (микросервисы[1]) — интересная идея. И вот не успели мы оглянуться, как они стали архитектурой, принятой по умолчанию в сотнях компаний по всему миру (многие, вероятно, запущены как стартапы, призванные решать проблемы, вызванные микрослужбами), что заставило всех "перейти на бег", чтобы успеть "запрыгнуть на подножку последнего вагона", который, как они опасаются, вот-вот исчезнет за горизонтом.

Должен признаться, здесь есть часть моей вины. С тех пор как в 2015 году я написал свою собственную книгу "Создание микросервисов " (Building Microservices) на эту тему, я зарабатываю на жизнь, работая с людьми, помогая им понять данный тип архитектуры. Я всегда пытался сделать одно — прорваться сквозь хайп и помочь компаниям определиться, подходят ли им микрослужбы или нет. Для многих моих клиентов с существующими (не ориентированными на микрослужбы) системами трудность состояла в том, как внедрить архитектуры, основанные на микрослужбах. Как взять существующую систему и выполнить перепланировку ее архитектуры, не останавливая всю остальную работу? Вот где на помощь приходит эта книга. Что еще важнее, я постараюсь дать вам честную оценку трудностей, связанных с архитектурой на основе микрослужб, и помочь вам понять, стоит ли начинать это "путешествие".

Чему вы научитесь

Эта книга задумана как глубокое погружение в образ мыслей и порядок действий при разложении существующих систем на архитектуру, основанную на микрослужбах. Мы коснемся многих тем, связанных с архитектурой на основе микрослужб, но в центре внимания будет находиться декомпозиция. В качестве более общего руководства по архитектуре на основе микрослужб хорошим местом для старта была бы моя предыдущая книга "Создание микросервисов ". На самом деле я настоятельно рекомендую вам рассматривать ту книгу как дополнение к этой.

Глава 1 содержит общий обзор того, что такое микрослужбы, и далее разведаны идеи, которые привели нас к такого рода архитектуре. Этот материал будет хорошим подспорьем для новичков в микрослужбах, но я также настоятельно призываю даже более опытных разработчиков не пропускать эту главу. Чувствую, что в шквале технологий некоторые стержневые идеи микрослужб часто упускаются из виду: к этим концепциям книга будет возвращаться снова и снова.

[1] Далее "микрослужбы", см. *Комментарии переводчика* (после *Предисловия*).

Глубоко разбираться в микрослужбах совсем неплохо, но знать, подходят они вам или нет, — это нечто другое. В *главе 2* я расскажу, как оценить пригодность микрослужб для ваших условий, а также дам некоторые действительно важные рекомендации по управлению транзитом с монолита на архитектуру, основанную на микрослужбах. Здесь мы коснемся всего, от доменно-обусловленного дизайна до моделей организационных изменений — жизненно важных основ, которые окажут вам неоценимую помощь, даже если вы решите не использовать архитектуру на основе микрослужб.

В *главах 3* и *4* мы глубже погрузимся в технические аспекты, связанные с декомпозицией монолита, изучая реальные примеры и выделяя шаблоны миграции. *Глава 3* сосредоточена на аспектах декомпозиции приложений, а *глава 4* представляет собой глубокое погружение в вопросы, связанные с данными. Если вы действительно хотите перейти с монолитной системы на архитектуру, основанную на микрослужбах, то вам придется разобрать некоторые базы данных на части!

Наконец, в *главе 5* рассматриваются всякого рода трудности, с которыми вы столкнетесь по мере роста архитектуры на основе микрослужб. Эти системы способны принести огромную выгоду, но они также сопровождаются большой сложностью и проблемами, с которыми вам не приходилось сталкиваться раньше. Эта глава является моей попыткой помочь вам засечь такие проблемы, когда они только начинают возникать, и предложить способы борьбы с болезнями роста, связанными с микрослужбами.

Условные обозначения, принятые в книге

В книге используются следующие типографские условные обозначения:

- *Курсив* — указывает новые термины, URL-адреса, адреса электронной почты, имена файлов и расширения файлов.
- `Моноширинный шрифт` — применяется для листингов программ, а также внутри абзацев для ссылки на элементы программ, такие, как переменные или имена, инструкции языка и ключевые слова.
- **`Полужирный моноширинный шрифт`** — выделяет команды либо другой текст, который должен быть напечатан пользователем буквально.
- *`Моноширинный шрифт курсивом`* — показывает текст, который должен быть заменен значениями пользователя либо значениями, определяемыми по контексту.

Данный элемент обозначает подсказку или совет.

Данный элемент обозначает общее замечание.

Данный элемент обозначает предупреждение или предостережение.

На веб-странице книги перечислены ошибки, приведены примеры и дополнительная информация. Вы можете обратиться к этой странице по адресу:

https://oreil.ly/monolith-to-microservices.

Благодарности

Без помощи и понимания моей замечательной жены Линди Стивенс эта книга была бы невозможной. Эта книга для нее. Линди, прости, что я так ворчал, когда приближались и проходили разные сроки завершения. Я также хотел бы поблагодарить прекрасный клан Гиллман Стейнс за всю их поддержку — мне повезло, что у меня такие замечательные родные.

Эта книга во многом выиграла благодаря людям, которые любезно пожертвовали свое время и энергию на то, чтобы прочитать различные черновики и дать ценные идеи. Я особенно хочу поблагодарить Криса О'Делла, Дэниела Брайанта, Пита Ходжсона, Мартина Фаулера, Стефана Шрасса и Дерека Хаммера за их усилия в этом деле. Были также люди, которые непосредственно внесли свой вклад во многих отношениях, поэтому я также хотел бы здесь поблагодарить Грэма Тэкли, Эрика Доэрненберга, Марчина Засепу, Майкла Фетера, Рэнди Шоуп, Кифа Моррис, Питера Гилларда-Мосса, Мэтта Хита, Стива Фримена, Рене Ленгвината, Сару Уэллс, Риса Эванса и Берка Сохана. Если вы найдете ошибки в этой книге, то эти ошибки будут моими, а не их.

Коллектив издательства O'Reilly также оказал невероятную поддержку, и я хотел бы высоко оценить труд моих редакторов Элеоноры Брю и Алисии Янг, в дополнение к Кристоферу Гузиковски, Мэри Трезелер и Рейчел Румелиотис. Также хочу сказать большое спасибо Хелен Кодлинг и ее коллегам по всему миру за то, что они продолжают таскать мои книги на различные конференции, Сьюзен Конант за то, что она помогает мне оставаться в здравом уме, ориентируясь в меняющемся мире издательского дела, и Майку Лукидесу за то, что он изначально привлек меня к работе с O'Reilly. Я знаю, что есть еще много людей "за кулисами", которые оказывали помощь, так что спасибо вам всем.

Помимо тех, кто непосредственно внес свой вклад в эту книгу, также хочу назвать и тех, кто осознанно или нет, помог этой книге появиться. Поэтому хотел бы поблагодарить (в произвольном порядке) Мартина Келппманна, Бена Стопфорда, Чарити Мейджорс, Алистера Кокберна, Грегора Хупе, Бобби Вулфа, Эрика Эванса, Ларри Константина, Лесли Лэмпорта, Эдварда Йордона, Дэвида Парнаса, Майка Блэнда, Дэвида Вудса, Джона Оллспоу, Альберто Брандолини, Фредерика Брукса, Синди Сридхаран, Дейва Фарли, Джез Хамбл, Джина Кима, Джеймса Льюиса, Николь Форсгрен, Гектора Гарсиа-Молину, Sheep & Cheese, Кеннет Салема, Адриан Кольера, Пэе Хелланд, Крестен Торупа, Хенрика Книберга, Андерса Иварссона, Мануэль Пайс, Стива Смита, Бернда Ракера, Мэтью Скелтона, Алексиса Ричардсона, Джеймса Говернера и Кейн Стивенс.

Как это всегда бывает в подобных ситуациях, мне кажется весьма вероятным, что я упустил кого-то, кто внес в эту книгу существенный вклад. Этим людям я могу ска-

зать только одно: мне очень жаль, что я забыл поблагодарить вас лично, и я надеюсь, что вы сможете простить меня.

Наконец, время от времени некоторые спрашивают меня об инструментах, которые мне потребовались для написания этой книги. Я писал в AsciiDoc, используя код Visual Studio вместе с плагином AsciiDoc, построенным Джоао Пинто (João Pinto). Книга находилась под контролем системы управления версиями Git, с системой Atlas O'Reilly. Я писал в основном на своем ноутбуке с внешней механической клавиатурой Razer, но ближе к концу также активно использовал iPad Pro с Git-клиентом Working Copy, для того чтобы закончить последние несколько разделов. Это позволило мне писать во время путешествия, обеспечив возможность в одном памятном случае писать о рефакторизации базы данных на пароме до Оркнейских островов. Вызванная этим морская болезнь стоила того.

Комментарии переводчика

Служба или сервис?

В настоящем переводе за основу принят широко известный методологический принцип "Бритвы Оккама", согласно которому следует избегать многообразия в терминологии без крайней на то необходимости (он формулируется и по-другому — не следует порождать "пустых" сущностей).

В ситуации, когда отсутствуют стандарты перевода научно-технических терминов, основными критериями служат авторитетные источники информации, каковыми являются Википедия[1], как унифицирующий (и нужно признать, иногда не бесспорный) источник знаний, и гиганты ИТ-индустрии.

Такие глобальные компании ИТ-индустрии, как Microsoft и IBM (их российские филиалы) в своей документации придерживаются перевода и интерпретации английского термина "microservice", именно как микрослужбы[2,3].

Микрослужбами называют тысячи независимых веб-стандартов, языков программирования, платформ баз данных и компонентов веб-серверов, используемых в современном жизненном цикле разработки ПО в качестве средств разработчиков. Организации, применявшие традиционный подход, предпочитали архитектуру, ориентированную на услуги, которая представляла собой сочетание оборудования и ПО, предоставляемых одной ИТ-компанией. Микрослужбы обеспечивают поддержку тысяч различных компонентов, предоставляемых независимыми компаниями по разработке или сообществами по созданию решений с открытым исходным кодом, в облачных приложениях и на веб-серверах. ИТ-отделам требовался новый подход к управлению микрослужбами в производственной инфраструктуре, состоящей из изолированных многоарендных сред в гипермасштабных центрах обработки данных (ЦОД) на базе публичных облаков. В связи с этим многие ИТ-отделы внедряли решения по виртуализации со стандартами программно-определяемого ЦОД на базе технологии Service Mesh. Микрослужбы представляют собой структурные блоки или основные компоненты, платформы и структуры, на базе которых создается и выполняется программный код на веб-серверах в облачном ЦОД.

---

[1] См. **https://ru.wikipedia.org/wiki/Веб-служба**.

[2] См. **https://docs.microsoft.com/ru-ru/dotnet/architecture/microservices**.

[3] См. **https://www.ibm.com/developerworks/ru/library/cl-bluemix-microservices-in-action-part-1-trs/**.

Следует добавить еще одну важную для понимания деталь. В информатике со времен ее формирования в 1960-х годах сложились определенные традиции перевода технических терминов, и с тех времен ничего не поменялось; изменились лишь люди, которые в последнее время все больше употребляют англицизмы, что очень часто затуманивает смысл и суть дела. Каждый термин принадлежит иерархии терминов, составляющей так называемое "древо знаний". Когда мы говорим "шаблон архитектурного дизайна" (pattern), мы четко определяем место этого термина в указанной иерархии. С другой стороны, перевод "паттерн" эту связь обрывает. То же касается, скажем, термина "каркас" (framework), который в зависимости от контекста, может быть вычислительным или математическим каркасом программирования и разработки. И, опять-таки, перевод "фреймворк" помимо своей непроизносимости на русском эту связь обрывает. Все указанное относится и к термину "microservice".

По всем этим причинам в настоящей книге термин "microservice" переведен как "микрослужба".

ГЛАВА 1

Основные сведения о микрослужбах

Скажем так, все обострилось в один миг и быстро вышло из-под контроля!
– *Рон Бургунди, Ведущий*

Прежде чем мы углубимся в приемы работы с микрослужбами, важно, чтобы у нас было общее, совместное понимание того, что такое архитектура на основе микрослужб. Я хотел бы обратиться к некоторым распространенным заблуждениям, которые встречаю регулярно, а также к нюансам, которые часто упускаются. Вам понадобится этот прочный фундамент знаний, чтобы извлечь максимальную пользу из остальной части книги. В данной главе приведено объяснение архитектуры на основе микрослужб, кратко рассказано о том, как микрослужбы развивались (что, естественно, означает рассмотрение монолитов), а также дан анализ некоторых преимуществ и трудностей работы с микрослужбами.

Что такое микрослужбы?

Микрослужбы — это независимо развертываемые службы, моделируемые вокруг бизнес-домена. Они общаются друг с другом через сети и предлагают на выбор целый ряд вариантов архитектуры для решения задач, с которыми вы можете столкнуться. Отсюда следует, что архитектура на основе микрослужб базируется на многочисленных сотрудничающих микрослужбах.

Рассматриваемый архитектурный тип ориентирован на службы (service-oriented architecture, SOA); не углубляясь в вопрос о том, как должны очерчиваться контуры служб, отметим, что ключом является их независимая развертываемость. Преимущество микрослужб также в том, что они нейтральны к технологиям.

С технологической точки зрения микрослужбы выставляют наружу возможности бизнеса, которые они инкапсулируют через одну или несколько конечных точек в сети. Микрослужбы взаимодействуют друг с другом через эти сети, что делает их разновидностью распределенной системы. Они также инкапсулируют хранение и извлечение данных, предоставляя данные через четко определенные интерфейсы. И поэтому базы данных скрыты в пределах контура службы.

Здесь есть много чего, в чем следует разобраться, поэтому давайте слегка углубимся в некоторые из этих идей.

Независимая развертываемость

Независимая развертываемость — это идея о том, чтобы вносить изменения в микрослужбу и развертывать ее в производственной среде без необходимости использовать какие-либо другие службы. Что еще важнее, дело не просто в том, что мы *можем* это осуществить, а в том, как вы управляете развертываниями в своей системе *на самом деле*. Это дисциплина, которую вы практикуете в большинстве ваших релизов. Это простая идея, которая, тем не менее, сложна в исполнении.

Если из этой книги вы извлечете только одну полезную вещь, то она должна быть такой: обеспечить внедрение концепции независимой развертываемости микрослужб. Приобретите привычку вносить изменения в свое производство в отдельной микрослужбе без развертывания чего-либо еще. Из этого последует много хорошего.

Для обеспечения гарантии независимой развертываемости наши службы должны быть слабо сопряжены — другими словами, мы должны иметь возможность изменять одну службу без необходимости менять что-либо еще. Это означает, что нам нужны ясные, четко определенные и стабильные контракты между службами. Некоторые варианты имплементации это затрудняют, например, использование совместных баз данных является особенно проблемным. Стремление к слабо сопряженным службам со стабильными интерфейсами направляет наше мышление прежде всего на отыскание контуров служб.

Моделируются вокруг бизнес-домена

Внесение изменения через контур процесса обходится дорого. Если для внедрения функции вам нужно изменить две службы и выполнить оркестровку указанных двух изменений, то это займет больше времени, чем внесение одинакового изменения внутри одной службы (или, если на то пошло, монолита). Из этого следует, что мы хотим обеспечить, чтобы трансграничные изменения между службами вносились нами как можно реже.

Следуя тому же подходу, который я употребил в книге "Создание микросервисов", в этой книге используется поддельный домен и фиктивная компания с целью иллюстрации некоторых концепций, когда невозможно поделиться реальными историями. Речь идет о компании Music Corp, крупной многонациональной организации, которая так или иначе остается в бизнесе, несмотря на то что она почти полностью сосредоточена на продаже компакт-дисков.

Упираясь руками и ногами, мы решили перенести Music Corp в XXI век, и в рамках этой компании мы оцениваем существующую системную архитектуру. На рис. 1.1 мы видим простую трехслойную архитектуру. У нас есть веб-интерфейс пользователя, слой бизнес-логики в форме монолитного бэкэнда (серверной части) и хранилища данных в традиционной базе данных. Эти слои, как обычно, принадлежат разным операционным группам.

Мы хотим внести в нашу функциональность простое изменение: нам нужно, чтобы наши заказчики могли указывать свой любимый жанр музыки. При этом потребу-

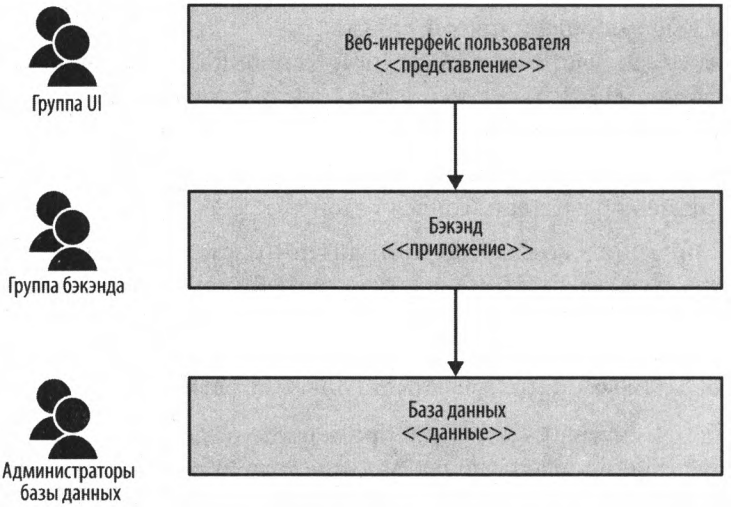

Рис. 1.1. Системы компании Music Corp как традиционная трехслойная архитектура

ется изменить пользовательский интерфейс (UI), показывая в нем жанры на выбор, чтобы бэкэнд обеспечивал появление жанра в UI и изменение значения и чтобы база данных принимала это изменение. Эти изменения должны управляться каждой группой, как показано на рис. 1.2, и они должны быть развернуты в правильном порядке.

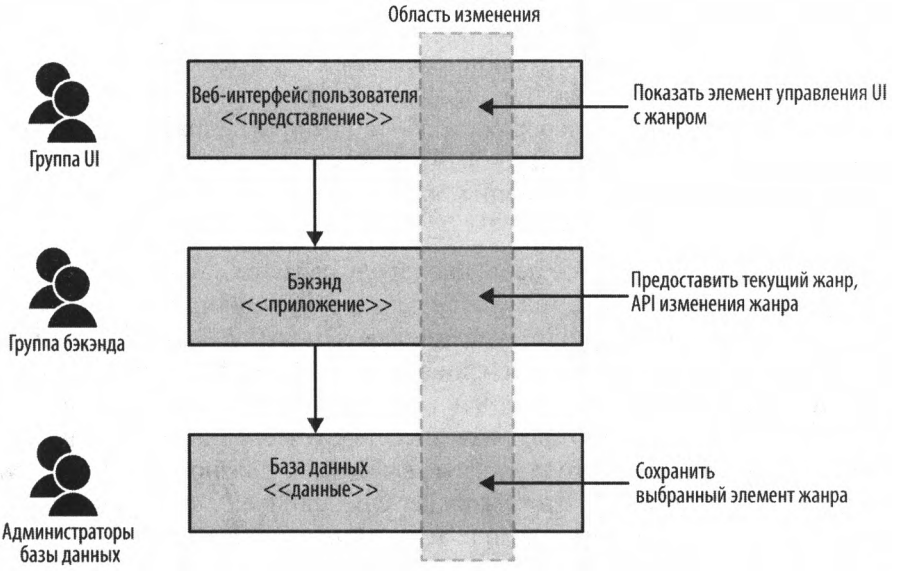

Рис. 1.2. Вносить изменения во всех трех слоях оказывается труднее

Теперь эта архитектура является не такой уж плохой. Вся архитектура в итоге оптимизируется вокруг некоторого множества целей. Трехъярусная архитектура настолько распространена отчасти потому, что она универсальна — все о ней слы-

шали. Так что выбор общепринятой архитектуры, которую вы, возможно, видели где-то в другом месте, часто является одной из причин, по которой мы продолжаем видеть этот шаблон. Но я думаю, что самая веская причина, почему мы встречаем эту архитектуру снова и снова, связана с тем, что она основана на способе организации наших групп разработчиков.

Ставший уже знаменитым закон Конвея гласит:

> "Любая организация, которая строит дизайн системы... неизбежно произведет дизайн, структура которого является копией коммуникационной структуры организации".
>
> – *Мелвин Конвей (Melvin Conway)*,
> Каким образом комитеты изобретают (How Do Committees Invent)?

Трехъярусная архитектура — хороший пример этого закона в действии. В прошлом ИТ-организации группировали людей в соответствии с их стержневыми компетенциями: администраторы баз данных — в группе с другими администраторами баз данных; разработчики на Java — в группе с другими разработчиками на Java; и разработчики фронтэнда (фронтальной части) (которые в настоящее время знают такие экзотические вещи, как JavaScript и разработка собственных мобильных приложений) были еще в одной группе. Мы группируем людей на основе их стержневых компетенций, поэтому мы создаем ИТ-активы, которые выравниваются по этим группам.

Указанный факт объясняет, почему эта архитектура так распространена. Она неплохая; она просто оптимизирована вокруг одного множества сил — так же как мы традиционно группируем людей — вокруг дружеских отношений. Но силы изменились. Наши стремления относительно программно-информационного обеспечения[1] изменились. Мы теперь объединяем людей в поликвалифицированные группы с целью сократить эстафетные передачи и обособленные подразделения. Мы хотим осуществлять доставку софта гораздо быстрее, чем когда-либо прежде. Это заставляет нас принимать различные решения, касающиеся организации наших групп и разделения наших систем на части.

Изменения в функциональности в первую очередь связаны с изменениями в бизнес-функциональности. Но на рис. 1.1 наша бизнес-функциональность фактически распределена по всем трем слоям, увеличивая вероятность того, что изменение функциональности приведет к пересечению слоев. В такой архитектуре мы имеем высокую связность родственных технологий, но низкую связность бизнес-функциональности. Если мы хотим упростить внесение изменений, то нам нужно изменить способ группирования кода — мы выбираем связность бизнес-функциональности, а не технологии. В итоге каждая служба может и не содержать смесь этих трех слоев, но это уже вопрос имплементации локальной службы.

---

[1] Понятия "софт" и "программно-информационное обеспечение" в переводе используются взаимозаменяемо. Последний термин используется именно в такой формулировке, исходя из зарубежной трактовки термина software, как "программ и операционной информации, необходимых компьютеру" (ср. **https://en.wikipedia.org/wiki/Software**), т. е. софт — это логика и данные, которыми оперирует компьютер — *Пер.*