1 Первый взгляд на микросервисы

В этой главе:

- понятие микросервиса и его основные отличительные признаки;
- плюсы и минусы микросервисов;
- пример совместной работы микросервисов при обслуживании запроса пользователя;
- использование веб-фреймворка Nancy в простом приложении.

В этой главе я расскажу вам о микросервисах и продемонстрирую, чем они интересны. Мы также рассмотрим шесть отличительных признаков микросервиса. Наконец, я познакомлю вас с двумя важнейшими из используемых в этой книге технологий: с основанным на платформе .NET веб-фреймворком Nancy и конвейером промежуточного ПО OWIN.

1.1. Что такое микросервис

Микросервис (microservice) — сервис, предоставляемый удаленным API для использования остальной системой, обладающий одной и только одной чрезвычайно узконаправленной функциональной возможностью. Например, рассмотрим систему управления складом. Если разбить ее возможности на составные части, получится следующий список.

1. Принимаем поступающий на склад товар.
2. Определяем, где должен храниться новый товар.
3. Рассчитываем маршруты размещения внутри склада, чтобы поместить товар в нужные хранилища.
4. Распределяем маршруты размещения товара между работниками склада.
5. Получаем заказы.
6. Рассчитываем маршруты изъятия товара со склада для составления заказов.
7. Распределяем маршруты изъятия товара между работниками склада.

Рассмотрим, как реализовать первую из этих возможностей — приемку прибывшего на склад товара — в качестве микросервиса. Мы назовем этот микросервис *Receive Stock («Принять товар»)*.

1. По HTTP поступает запрос на приемку нового товара и занесение его в журнал. Он может исходить от другого микросервиса или, возможно, от веб-страницы, используемой диспетчером для регистрации поступившего товара. Микросервис Receive Stock должен зарегистрировать новый товар в собственном хранилище данных.
2. Микросервис отправляет ответ в качестве подтверждения приемки товара.

Рисунок 1.1 демонстрирует получение микросервисом Receive Stock запроса от другого микросервиса, работающего совместно с ним.

Рис. 1.1. Микросервис Receive Stock при поступлении нового товара предоставляет для использования API, к которому могут обращаться другие микросервисы

Каждая маленькая возможность в системе реализуется в виде отдельного микросервиса. Любой микросервис в системе:

- работает в собственном отдельном процессе;
- может быть развернут отдельно, независимо от других микросервисов;
- имеет собственное хранилище данных;
- взаимодействует с другими микросервисами для выполнения своих заданий.

Важно также отметить, что микросервисы могут взаимодействовать с другими микросервисами, не обязательно написанными на том же языке программирования (C#, Java, Erlang и т. д.). Все, что им требуется знать, — как обмениваться сообщениями друг с другом. Некоторые могут обмениваться сообщениями через служебную шину или двоичный протокол, например Thrift, в зависимости от системных требований, но гораздо чаще микросервисы обмениваются сообщениями по протоколу HTTP.

ПРИМЕЧАНИЕ

Эта книга посвящена созданию микросервисов на платформе .NET с помощью языка программирования C# и веб-фреймворка Nancy. Микросервисы, которые я буду демонстрировать, представляют собой маленькие узкоспециализированные приложения Nancy, взаимодействующие по HTTP.

Какова архитектура микросервисов

Основное внимание в этой книге уделяется проектированию и реализации отдельных микросервисов, но стоит отметить, что термин *«микросервисы»* можно использовать также для описания *архитектурного стиля* всей системы, состоящей из множества микросервисов. Микросервисы как архитектурный стиль — упрощенный вариант сервис-ориентированной архитектуры (service-oriented architecture (SOA)), в которой сервисы ориентированы на выполнение одного действия каждый, и выполнения качественного. Система с архитектурой микросервисов представляет собой распределенную систему с, вероятно, значительным количеством совместно работающих микросервисов.

Микросервисный архитектурный стиль быстро обретает все бóльшую популярность при создании и поддержке сложных серверных комплексов программ. Причина этого очевидна: микросервисы обеспечивают множество потенциальных преимуществ по сравнению как с более традиционными сервис-ориентированными подходами, так и с монолитными архитектурами. Микросервисы при надлежащей реализации демонстрируют пластичность, масштабируемость и отказоустойчивость вдобавок к небольшой длительности производственного цикла от начала создания до внедрения в производство.

Отличительные признаки микросервисов

Как я упоминал, микросервис — это *сервис с одной узконаправленной функциональной возможностью*. Что эта фраза значит? Поскольку методология микросервисов все еще находится в стадии развития (по состоянию на начало 2016 года), пока не существует общепринятого определения того, что именно представляет собой микросервис[1]. Можно, однако, посмотреть, что отличает микросервис в целом. Я обнаружил, что существует шесть основных отличительных признаков микросервиса.

1. Микросервис отвечает за одну функциональную возможность.
2. Микросервисы можно развертывать по отдельности.
3. Микросервис состоит из одного или нескольких процессов.
4. У микросервиса имеется собственное хранилище данных.
5. Небольшая команда разработчиков может сопровождать несколько микросервисов.
6. Микросервис можно легко заменить.

Этот список отличительных признаков поможет вам при столкновении с правильно построенным микросервисом распознать его, а также определить сферу действия и реализации своих собственных микросервисов. Приняв за основу эти

[1] Для дальнейшего обсуждения отличительных признаков микросервисов я рекомендовал бы вам обратиться к статье на эту тему: *Fowler M., Lewis J.* Microservices: A Definition of This New Architectural Term, 2014. — March 25 [Электронный ресурс]. — Режим доступа: http://martinfowler.com/articles/microservices.html.

отличительные признаки, вы вступите на путь извлечения максимальной выгоды из своих микросервисов и создания в результате *пластичной*, *масштабируемой* и *отказоустойчивой* системы. На протяжении всей книги я буду демонстрировать, что эти отличительные признаки означают в смысле устройства микросервисов и как писать необходимый для их воплощения код. Теперь рассмотрим каждый из признаков по отдельности.

Ответственность за одну возможность

Микросервис *отвечает за одну и только одну функциональную возможность* из всей системы. Разделим это утверждение на две составные части.

- На микросервис возлагается ровно одна обязанность.
- Эта обязанность состоит в реализации отдельной возможности.

Принцип единственной обязанности формулируется несколькими различными способами. Одна из традиционных его формулировок: «У класса должна быть только одна причина для изменения»[1]. Хотя здесь говорится именно о *классе*, этот принцип может применяться и вне контекста классов в объектно-ориентированных языках программирования. В случае микросервисов принцип единственной обязанности применяется на уровне сервисов.

Более современная формулировка принципа единственной обязанности тоже обязана своим появлением дядюшке Бобу: «Собирайте воедино вещи, меняющиеся по схожим причинам. Отделяйте друг от друга вещи, меняющиеся по различным причинам»[2]. Эта формулировка данного принципа применима к микросервисам: микросервис должен реализовывать ровно одну функциональную возможность. Таким образом, микросервис должен изменяться только при изменении этой возможности. Более того, вы должны постараться реализовать в микросервисе эту возможность максимально полно, чтобы при ее изменении нужно было менять только один микросервис.

Существует два вида возможностей в системе микросервисов.

- *Бизнес-возможность* — выполняемое системой действие, которое способствует реализации предназначения системы. Например, это может быть отслеживание корзин заказов пользователей или расчет цен. Предметно-ориентированное проектирование — отличный способ разделения системы на отдельные бизнес-возможности.
- *Техническая возможность* — нечто необходимое для нескольких других микросервисов, например для интеграции с какой-то сторонней системой. Технические возможности нельзя назвать главными движущими силами при разбиении системы на микросервисы, они выделяются лишь в случае, когда оказывается, что нескольким микросервисам для их бизнес-возможностей требуется одна и та же техническая возможность.

---

1 *Martin R. C.* SRP: The Single Responsibility Principle [Электронный ресурс]. — Режим доступа: http://mng.bz/zQyz.

2 *Martin R. C.* The Single Responsibility Principle [Электронный ресурс]. — Режим доступа: http://mng.bz/RZgU.

ПРИМЕЧАНИЕ

Определение области действия и обязанностей микросервисов рассматривается в главе 3.

Развертываемость по отдельности

Микросервисы должны допускать *развертываемость по отдельности*. При изменении микросервиса должна существовать возможность ввести этот измененный микросервис в эксплуатацию, не развертывая (вообще не затрагивая) какие-либо другие части системы. Остальные микросервисы в системе должны выполняться и работать во время развертывания измененного микросервиса и продолжать выполняться после того, как новая версия будет развернута.

Рассмотрим сайт интернет-магазина. При изменении микросервиса Shopping Cart («Корзина заказов») должна быть обеспечена возможность развертывания только этого микросервиса (рис. 1.2).

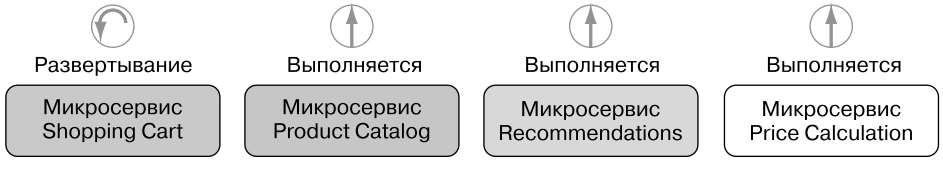

Рис. 1.2. Во время развертывания микросервиса Shopping Cart остальные микросервисы продолжают работать

Тем временем микросервисы Price Calculation («Расчет цены»), Recommendations («Рекомендации»), Product Catalog («Каталог товаров») и другие продолжают работать и обслуживать запросы пользователей.

Возможность индивидуального развертывания каждого из микросервисов важна, поскольку в системе их может быть очень много и каждый может взаимодействовать с несколькими другими. Одновременно с этим выполняется развертывание некоторых или всех микросервисов. Если бы пришлось развертывать все микросервисы или их группы в строгом порядке, управление развертываниями быстро стало бы громоздким, что привело бы к нечастым и объемным рискованным развертываниям. Несомненно, вместо этого желательно иметь возможность частого развертывания небольших изменений во всех микросервисах, что означает небольшие развертывания с низким уровнем риска.

Чтобы иметь возможность развертывать отдельный микросервис в тот момент, когда остальная система продолжает работать, процесс сборки должен быть настроен с учетом следующего.

- Все микросервисы должны быть встроены в отдельные артефакты или пакеты.
- Процесс развертывания должен быть настроен таким образом, чтобы обеспечить поддержку индивидуального развертывания при продолжении функционирования остальных микросервисов. Например, можно использовать процесс плавающего развертывания, при котором микросервис развертывается на одном сервере за один раз, чтобы снизить время простоя.

То, что микросервисы желательно развертывать по отдельности, влияет на способ их взаимодействия. Изменения в интерфейсе микросервиса обычно должны быть обратно совместимыми, чтобы другие микросервисы могли продолжать работать с новой версией точно так же, как работали со старой. Более того, способ взаимодействия микросервисов должен быть устойчивым к ошибкам в том смысле, что каждый микросервис должен быть готов к периодическим сбоям других микросервисов и должен при этом продолжать работать по мере возможности. Сбой одного микросервиса, например, из-за простоя во время развертывания не должен приводить к сбою других микросервисов, а лишь к сокращению функциональности или чуть более длительному времени обработки.

ПРИМЕЧАНИЕ

Взаимодействие и устойчивость к ошибкам микросервисов мы рассмотрим в главах 4, 5 и 7.

Состоит из одного или нескольких процессов

Микросервис должен выполняться в отдельном процессе или отдельных процессах, чтобы оставаться как можно более независимым от других микросервисов той же системы, а также чтобы сохранять возможность отдельного развертывания. Разбив это положение на составные части, получаем два пункта.

- Все микросервисы должны работать в отдельных от других микросервисов процессах.
- У любого микросервиса может быть более одного процесса.

Вернемся к микросервису Shopping Cart. Если он работает в том же процессе, что и микросервис Product Catalog (рис. 1.3), код микросервиса Shopping Cart может вызывать побочные эффекты в микросервисе Product Catalog. Это означает сильную (и нежелательную) связанность между микросервисами Shopping Cart и Product Catalog: работа одного может приводить к простоям или ошибкам другого.

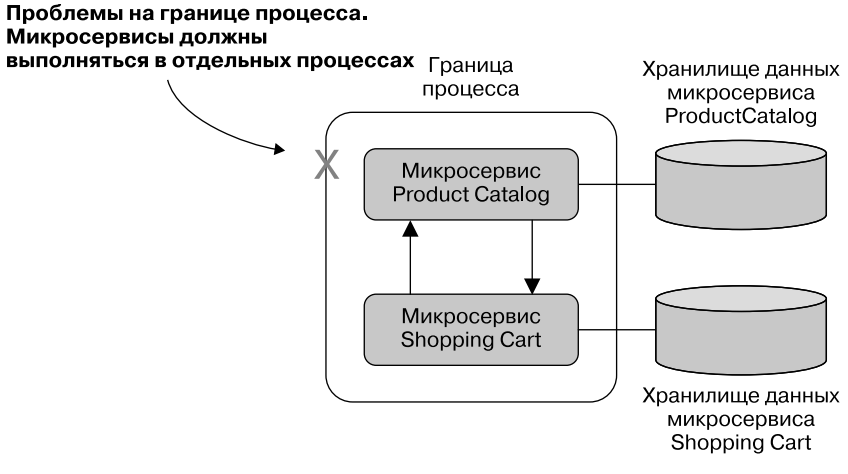

Рис. 1.3. Выполнение нескольких микросервисов в одном процессе ведет к сильной связанности

Теперь рассмотрим развертывание новой версии микросервиса Shopping Cart. Оно или повлечет за собой повторное развертывание микросервиса Product Catalog, или потребует какой-нибудь разновидности динамической загрузки кода, которая бы позволила заменить код микросервиса Shopping Cart в работающем процессе. Первый вариант напрямую противоречит положению об индивидуальной развертываемости микросервисов. Второй же сложен и как минимум ставит микросервис Product Catalog под угрозу сбоя вследствие развертывания микросервиса Shopping Cart.

Говоря о сложности: почему микросервис может состоять более чем из одного процесса? Мы ведь в конце концов стремимся к максимальному упрощению микросервисов.

Рассмотрим микросервис Recommendations. Он реализует и выполняет алгоритмы выдачи рекомендаций для сайта интернет-магазина, а также содержит базу данных, в которой хранятся необходимые для этого данные. Алгоритмы выполняются в одном процессе, а база данных работает в другом. Зачастую микросервису необходимо два или более процесса для реализации всего того (например, хранилища данных и фоновой обработки), что требуется ему для обеспечения системе соответствующей функциональной возможности.

Наличие собственного хранилища данных

У микросервиса должно быть собственное хранилище, в котором находятся необходимые ему данные. Это еще одно следствие того, что сфера действия микросервиса — завершенная функциональная возможность. Большей части бизнес-возможностей требуется какое-либо хранилище данных. Например, микросервису Product Catalog необходимо хранить определенную информацию о каждом из товаров. Сохранение слабой связанности микросервиса Product Catalog с другими микросервисами обеспечивается тем, что содержащее информацию о товарах хранилище данных полностью принадлежит ему. Только микросервис Product Catalog определяет, как и когда сохраняется информация о товарах. Как показано на рис. 1.4, другие микросервисы, например микросервис Shopping Cart, могут обращаться к информации о товарах только через интерфейс микросервиса Product Catalog, а не напрямую в хранилище данных Product Catalog.

То, что у каждого микросервиса имеется собственное хранилище данных, позволяет использовать различные технологии баз данных для различных микросервисов в зависимости от потребностей конкретного микросервиса. Например, микросервис Product Catalog может использовать для хранения информации о товарах СУБД SQL Server, микросервис Shopping Cart — хранить содержимое корзин заказов всех пользователей в СУБД Redis, а микросервис Recommendations — использовать для выдачи рекомендаций индекс Elasticsearch. Выбираемые для микросервиса технологии баз данных — часть реализации, невидимая другим микросервисам.

Такой подход дает каждому микросервису возможность использовать оптимальную для его задач базу данных, что приносит выигрыш в смысле времени разработки, производительности и масштабируемости. Очевидный недостаток — необходимость администрирования и сопровождения нескольких баз данных, а также работы с ними, если вы спроектировали свою систему подобным образом. Базы данных часто оказываются сложными технологически, и научиться использовать каждую

из них на уровне эксплуатации в производственной среде непросто. Следует принимать во внимание это соотношение выгод и потерь при выборе базы данных для микросервиса. Но одно из преимуществ наличия у каждого микросервиса своего хранилища данных — возможность замены одной базы данных на другую.

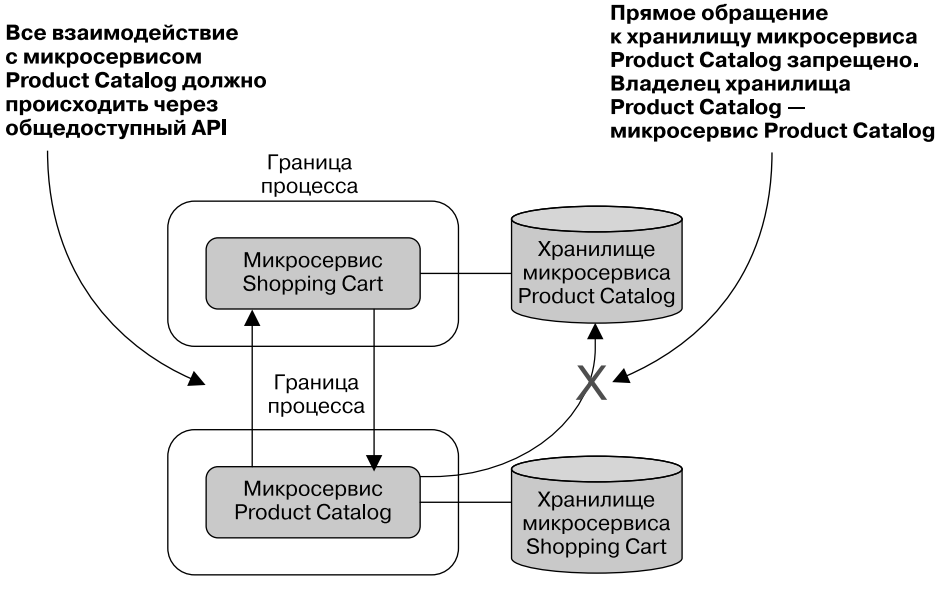

Рис. 1.4. Один микросервис не может обращаться к хранилищу другого

ПРИМЕЧАНИЕ

Мы рассмотрим владение данными, доступ к ним и их хранение в главе 5.

Возможность сопровождения небольшой командой разработчиков

До сих пор я практически не упоминал о размере микросервиса, хотя корень «микро» говорит о том, что микросервисы невелики. Я не думаю, что имеет смысл обсуждать, сколько строк кода должно быть в микросервисе или каково количество реализуемых им технических требований, сценариев использования или функциональных точек. Все это зависит от сложности обеспечиваемой микросервисом функциональной возможности.

А вот обсудить объем работ по сопровождению микросервиса имеет смысл. В качестве руководящего принципа при выборе размера микросервисов можно использовать следующее эмпирическое правило: *небольшая команда разработчиков — скажем, пятеро — должна быть в состоянии сопровождать как минимум пять микросервисов*. Термин *«сопровождать»* означает все аспекты поддержания работоспособности микросервиса и его пригодности для использования: разработку новой функциональности, выделение новых микросервисов из слишком разросшихся старых, обслуживание при эксплуатации в производственной среде, мониторинг, тестирование, исправление ошибок — и все, что только потребуется.

Заменяемость

Заменяемость микросервиса означает возможность переписать его с нуля за разумное время. Другими словами, *сопровождающая микросервис команда должна быть способна заменить текущую реализацию совершенно новой, причем сделать это в обычном рабочем режиме*. Этот отличительный признак накладывает еще одно ограничение на размер микросервиса: замена слишком большого микросервиса обойдется недешево, а заменить маленький вполне реально.

По какой же причине может возникнуть необходимость замены микросервиса? Возможно, код пришел в беспорядок и сопровождать его стало слишком сложно. Может быть, на производстве он демонстрирует не слишком хорошую производительность. Обе ситуации нежелательны, но изменения в технических требованиях с течением времени часто приводят к тому, что имеет смысл заменить код, а не поддерживать. Если размер микросервиса достаточно мал для того, чтобы переписать его за разумный промежуток времени, в периодическом появлении таких ситуаций нет ничего страшного. Команда разработчиков переписывает его, учитывая опыт создания текущей реализации и всех новых технических требований.

Мы познакомились с отличительными признаками микросервисов. Обсудим теперь выгоды от их использования, затраты и другие соображения.

1.2. Почему именно микросервисы?

Создание системы микросервисов, соответствующих описанным в предыдущем разделе принципам, несет очевидные выгоды: они пластичны, масштабируемы и отказоустойчивы, и время от начала их разработки до внедрения в производство невелико. Эти выгоды реальны, поскольку надлежащим образом созданные микросервисы:

- обеспечивают возможность непрерывной доставки ПО;
- обеспечивают эффективность процесса разработки в силу чрезвычайной легкости их сопровождения;
- изначально выполнены устойчивыми к ошибкам;
- масштабируются в сторону увеличения или уменьшения независимо друг от друга.

Обсудим эти пункты подобнее.

Возможность непрерывной доставки ПО

Архитектурный стиль микросервисов учитывает необходимость непрерывной доставки ПО. Это делается благодаря акценту на сервисах, которые:

- можно быстро разработать и изменить;
- можно всесторонне тестировать с помощью автоматизированных тестов;
- можно развертывать независимо друг от друга;
- работают эффективно.

Эти свойства делают возможной непрерывную доставку ПО, но непрерывная доставка отнюдь не вытекает из использования архитектуры микросервисов. Их взаимосвязь носит более сложный характер: реализация непрерывной доставки упрощается благодаря микросервисам по сравнению с более традиционной сервис-ориентированной архитектурой. В то же время только при надежном и эффективном развертывании сервисов появляется возможность полностью реализовать концепцию микросервисов. Непрерывная доставка ПО и микросервисы взаимодополняют друг друга.

Выгоды от непрерывной доставки ПО хорошо известны. Это повышенная адаптивность на бизнес-уровне, надежность выпуска ПО, снижение рисков и повышение качества программного продукта.

Что такое непрерывная доставка ПО

Непрерывная доставка ПО — применяемая при разработке практика обеспечения возможности быстрого развертывания программного обеспечения в производственной среде в любой момент. Развертывание в производственной среде остается бизнес-решением, но практикующие непрерывную доставку команды предпочитают выполнять его часто и развертывать свежеразработанное программное обеспечение вскоре после попадания его в систему контроля исходных кодов.

Существуют два основных требования к непрерывной доставке. Во-первых, программное обеспечение всегда должно находиться в полностью рабочем состоянии. Чтобы добиться этого, команда должна уделять особое внимание контролю качества. Это ведет к повышению уровня автоматизации тестов и разбиению разработки на небольшие куски. Во-вторых, процесс развертывания должен быть надежным, повторяемым и быстрым, чтобы можно было часто развертывать программное обеспечение в производственной среде. Это достигается полной автоматизацией процесса развертывания и высокой степенью мониторинга состояния производственной среды.

Хотя непрерывная доставка ПО требует немалых технических навыков, это в большей степени вопрос организации процесса и культуры труда. Подобный уровень качества, автоматизации и мониторинга требует тесного сотрудничества между всеми вовлеченными в разработку и эксплуатацию программного обеспечения сторонами, включая предпринимателей, разработчиков, экспертов по информационной безопасности и системных администраторов. Другими словами, он требует культуры DevOps, когда те, кто занимается разработкой и эксплуатацией, сотрудничают и учатся друг у друга.

Непрерывная доставка ПО — неотъемлемый спутник микросервисов. При невозможности быстрого и дешевого развертывания отдельных микросервисов реализация системы микросервисов быстро стала бы очень затратной. В отсутствие автоматизации развертывания микросервисов количество необходимой при развертывании полной системы микросервисов ручной работы было бы ошеломляющим.

Непрерывной доставке сопутствует культура DevOps, также необходимая для микросервисов. Для успешной работы системы микросервисов нужно, чтобы каждый участник процесса вносил свой вклад в обеспечение беспроблемной работы сервиса и максимальной прозрачности состояния производственной среды. Это тре-

бует сотрудничества различных специалистов: с опытом эксплуатации, опытом разработки, опытом работы в сфере безопасности, а также со знанием предметной области, помимо прочего.

Эта книга не посвящена непрерывной доставке ПО или DevOps, но мы предполагаем, что в среде, в которой вы разрабатываете микросервисы, используется непрерывная доставка. Создаваемые в данной книге сервисы можно развертывать в локальных дата-центрах или в облаке с применением любого количества технологий автоматизации развертывания, допускающих использование платформы .NET. Эта книга охватывает последствия применения непрерывной доставки ПО и DevOps для отдельных микросервисов. В части III книги мы подробнее рассмотрим создание платформы, принимающей на себя немало эксплуатационных проблем, с которыми приходится иметь дело всем микросервисам. Кроме того, в приложении Б рассмотрим основные возможности ввода в производственную среду разработанных на протяжении всей книги микросервисов.

Высокий уровень удобства сопровождения

Грамотно спроектированные и разработанные микросервисы легки в сопровождении с нескольких точек зрения. *С точки зрения разработчика*, в обеспечении удобства сопровождения микросервисов играют роль несколько факторов.

- Каждый грамотно спроектированный микросервис обеспечивает *единственную функциональную возможность*. Не две и не три — только одну.
- У микросервиса имеется собственное хранилище данных. Никакие другие сервисы не могут взаимодействовать с хранилищем данных микросервиса. Этот факт, а также типичный объем кода микросервиса означают, что можно охватить весь микросервис одним взглядом и сразу понять, что он собой представляет.
- Для грамотно написанных микросервисов могут и должны быть созданы всесторонние автоматизированные тесты.

С точки зрения эксплуатации в обеспечении удобства сопровождения микросервисов играют роль два фактора.

- Небольшая команда разработчиков может сопровождать несколько микросервисов. Микросервисы должны быть спроектированы в расчете на эффективную эксплуатацию, из чего следует, что должна существовать возможность легко определять текущее состояние любого микросервиса.
- Все микросервисы можно разворачивать по отдельности.

Из этого следует, что необходимо своевременно обнаруживать и решать возникающие при эксплуатации в производственной среде проблемы, например, масштабированием соответствующего микросервиса или развертыванием его новой версии. То, что у микросервиса в соответствии с одним из отличительных признаков имеется собственное хранилище данных, также повышает удобство его сопровождения при эксплуатации в производственной среде, поскольку область сопровождения хранилища данных ограничивается его микросервисом-владельцем.