Краткое содержание

Оглавление

Часть II. Веб-сервисы

Часть III. Интеграция данных

Часть IV. Промышленная эксплуатация

Посвящается Еве и Макани.

С любовью от дяди Джоша

Посвящается моему дедушке Аббасу, который ждал появления своего имени на обложке книги целых сто лет.

Кенни

Предисловие Джеймса Уоттерса

В одну и ту же реку невозможно войти дважды.

Гераклит

Когда летом 2015 года я сидел с Джошем Лонгом в кафе на Венис-Бич, у меня появилось ощущение, что мы находимся на пороге чего-то большого. В то время наша платформа, ориентированная на работу в облачной среде, Pivotal Cloud Foundry, быстро набирала популярность в качестве среды выполнения облачных приложений, и разработчики очень хотели побольше узнать о нашей новой технологии — Spring Boot. В сочетании с набиравшей популярность Spring Boot выход на сцену Spring Cloud обещал произвести настоящий фурор. Я заметил: «Они будут прекрасно сочетаться, и это уже происходит».

К работе были привлечены неимоверные силы. В тот самый момент, когда ИТ-директора отчаянно искали пути повышения производительности труда проектировщиков, среда Spring Boot предлагала микросервисы и вполне приемлемый подход DevOps к разработке корпоративных приложений. Развивающаяся интеграция Spring Boot и PCF превратила развертывание в производственной среде в простой конвейер, API был отозван, Spring Cloud поставила первую в мире сеть микросервисов, и появился стандартный облачный подход к Java.

Изменение в порядке разработки, которое совсем не казалось поверхностным, уникальная комбинация этих технологий изменили структуру поставки продукта большими организациями. Слишком долго разработчикам не удавалось выполнять развертывание в производственной среде из-за эксплуатационной сложности устаревших серверов приложений и шаблонов Java. Нередко можно было услышать мрачные клиентские истории о многодневных развертываниях. Мы знали, что наша платформа изменит жизнь клиентов. После внедрения Spring Boot на PCF они

стали присылать нам благодарственные письма с описаниями обновлений в производственной среде, занимающих минуты, а не месяцы.

С 2015 года мы наблюдаем отличные результаты, и организации, перешедшие на эту технологию, отмечают как минимум 50-процентное повышение скорости создания программ, более чем двойное улучшение показателей наработки на отказ и времени простоя при восстановлении после сбоев, а также возможность управления десятками тысяч JVM-машин с небольшими командами обслуживания платформы. Самое главное, что организации, внедрившие технологии, рассмотренные в данной книге, больше времени уделяют решению вопросов, связанных с клиентами и рынками, и существенно меньше времени тратят на переживания по поводу сложностей разработки и сопровождения программ.

Страница за страницей эта книга подробно описывает наиболее важные модели современной разработки корпоративного программного обеспечения (ПО). С большим трудом накопленный Джошем и Кенни практический опыт взаимодействия со многими ведущими предприятиями в сфере производства сквозит во множестве приведенных здесь примеров.

Я призываю каждого проектировщика и ИТ-руководителя, готового воспользоваться всей мощью адаптивности и гибкости в своих организациях, получить удовлетворение от этой работы.

Джеймс Уоттерс (James Watters),
первый вице-президент Pivotal Cloud.
Foundry, Pivotal, @wattersjames

Предисловие Рода Джонсона

Мы становимся живыми свидетелями одного из самых перспективных преобразований в истории нашей отрасли: перехода от старых архитектур к облаку и от традиционного разделения на разработчиков и операторов к унифицированному понятию DevOps. Задача преобразования решается в данной книге путем объяснения возможностей и проблем написания приложений, ориентированных на работу в облачной среде, и предоставления четкого руководства относительно того, как это делается.

Преобразования не происходят в одночасье. Одна из самых сильных сторон данной книги — ее акцентированность на том, как добраться до облака оттуда, где вы оказались сегодня, опираясь на имеющийся опыт. В частности, в разделе о достижении паритета сервисов предоставляется замечательный материал о том, как перейти от традиционных методов работы к облачным.

Эта книга предлагает выверенный баланс теории и практики, объясняя и принципы построения архитектуры современных приложений, и эффективные, опробованные способы ее реализации. Практика требует выбора не только языка программирования, но и первичной среды с открытым кодом, поскольку современные приложения неизменно создаются с опорой на оправдавшие себя решения, имеющие открытый код. Если ваш выбор пал на язык Java — данная книга для вас.

Слухи о моей смерти сильно преувеличены.

Марк Твен

Несколько лет назад сообщения о «смерти» Java были обычным явлением. Сегодня же Java процветает, и эта книга напомнит вам причину данного явления. Язык

Java обрел новую жизнь отчасти потому, что Java-технологии привели к созданию современных приложений, готовых к работе в облачной среде. Двумя решающими факторами стали проекты с открытым кодом Netflix и Spring. В этой книге проделана замечательная работа по охвату обоих факторов.

Первоначально задуманные для упрощения сложностей Java EE минувшей эпохи, основные идеи Spring выдержали испытание временем и отлично зарекомендовали себя применительно к облачным приложениям. Более десяти лет назад мы говорили о Spring-треугольнике: внедрении зависимостей, абстракции переносимых сервисов и AOP. Все это в равной степени актуально и сегодня, когда чистое разобщение бизнес-логики и ее среды стало важнее, чем когда-либо.

Основа данной книги — освещение Spring Boot, нового способа использования Spring в эпоху микросервисов, который сегодня активно внедряется на производствах. Spring Boot упрощает создание новых сервисов Spring любой степени детализации и их развертывание в современной контейнеризированной среде. В то время как традиционное «корпоративное» Java-приложение было монолитным, запускаемым на все еще большом сервере приложений, среда Spring Boot развернула все в сторону упрощения и повышения эффективности: сервис должным образом становится центральной фигурой и развертывается с сервером, обладающим ресурсами, которых вполне хватает для его запуска.

В данной книге рассматривается последняя работа команды Spring — Spring Cloud Stream и усовершенствованная поддержка комплексного тестирования, а также развивающаяся интеграция с проектами Netflix с открытым кодом.

Я рад отметить продолжение нововведений Spring и концентрацию этой среды на упрощении труда проектировщиков. Хотя последние пять лет я работал со средой Spring только в качестве пользователя, мне радостно видеть, как она процветает и побеждает все новые источники сложности. Сегодня я продолжаю решение этой задачи упрощения в среде Atomist, где мы стремимся сделать для команд разработчиков и процесса создания ПО все то, что Spring сделала для приложений Java, предоставляя возможность автоматизировать все важные процессы. Среда Spring обеспечивает простую производительную структуру и абстракцию для работы со всем, чем занимается Java-разработчик. Среда Atomist нацелена на решение тех же задач в отношении исходного кода проекта, построения систем, отслеживания проблем, развернутой среды и т. д. Сюда же включена мощная автоматизация разработки: от создания новых проектов до усовершенствования сотрудничества команд с помощью Slack и отслеживания событий развертывания.

Фундаментальный строительный блок автоматизации — тестирование. В данной книге мне особенно понравилось всестороннее освещение этого процесса, проливающее свет на сложные проблемы, которые касаются тестирования микросервисов. Мне также понравилось наличие большого количества листингов программ, снабженных подробными комментариями.

Приятно писать предисловие для книги, сотворенной друзьями и посвященной любимой мною технологии. Те, кто знаком с Джошем, знают, что он отличный собеседник. Он так же хорош во владении печатным словом, как и в живом программировании. Джош и Кенни — страстные, любознательные и хорошо информированные экскурсоводы, и я с удовольствием путешествую с ними. Я многому научился на этом пути и уверен, что и вам это удастся.

Род Джонсон (Rod Johnson),
создатель, генеральный директор Spring Framework,
Atomist, @springrod

Введение

Быстрее! Быстрее! Быстрее!!! Всем хочется двигаться быстрее, но не все знают, как этого добиться. Требования рынка растут все быстрее, появляются и новые возможности, но некоторые из нас просто не в состоянии идти в ногу со временем! Чем отличается обычное предприятие от таких, как Amazon, Netflix и Etsy? Нам известно, что эти компании разрослись до неимоверных масштабов и все же каким-то образом сохраняют свое преимущество, опережая конкурентов. Как?

Довести идею от концепции до клиента, понять, что идея превратилась во что-то полезное и ценное, можно, проделав большой объем работы. На своем пути к вводу в эксплуатацию любой продукт проходит через множество различных этапов: доведение пользовательского восприятия до проектировщиков, разработка, тестирование, запуск в производство. Исторически сложилось так, что работа замедлялась на каждом из этих пунктов. Со временем общими усилиями мы оптимизировали различные части процесса. У нас имеются облачные вычисления, поэтому отпала необходимость в многоярусных сервисах. Мы используем разработку на основе тестирования и непрерывную интеграцию, предназначенную для автоматизации проверок. Программы выпускаются в небольших пакетах — микросервисах, что позволяет сократить область изменений и их стоимость. Мы принимаем идеи, заложенные в devops, чтобы способствовать всестороннему охвату системы и чувству товарищества между разработчиками и операторами, сокращая тем самым неоправданные затраты, связанные с рассогласованностью приоритетов. Все собранное вместе и есть то, что мы подразумеваем под *облачными технологиями*.

Разработчики программных продуктов, являясь специалистами и практиками в своей отрасли, оказались сегодня на перепутье. Существуют надежные, стабильные самообслуживающиеся решения с открытым кодом для инфраструктуры, тестирования, связки, непрерывной интеграции и поставки, сред разработки и облачных платформ. Эти элементы позволяют организациям сосредоточиться на менее затратной поставке ценностей более высокого порядка и в более широком масштабе.

Для кого эта книга

В основном данная книга предназначена для разработчиков Java- и JVM-машин, которые ищут способы создания более качественного ПО в короткие сроки с помощью Spring Boot, Spring Cloud и Cloud Foundry. Она для тех, кто уже слышал шум, поднявшийся вокруг микросервисов. Возможно, вы уже поняли, на какую стратосферную высоту взлетела среда Spring Boot, и удивляетесь тому, что сегодня предприятия используют платформу Cloud Foundry. Если так и есть, то эта книга для вас.

Зачем мы ее написали

В компании Pivotal мы помогаем клиентам превращать их компании в передовые организации цифровых технологий, обучая их непрерывной поставке и применению Cloud Foundry, Spring Boot и Spring Cloud. Мы видим, что срабатывает (а что — нет), и хотим зафиксировать состояние дел, как это определено нашими пользователями, и поделиться своим опытом. Мы не претендуем на всесторонний охват, но при этом стараемся затронуть ключевые понятия — те области мира облачных вычислений, в которых вы собираетесь работать, и дать о них ясное представление.

Структура книги

Книга имеет следующую структуру:

- в главах 1 и 2 рассказывается, чем хорошо «облачное» мышление, а затем предоставляется краткий экскурс в Spring Boot и Cloud Foundry;
- в главе 3 представлены способы конфигурации приложения Spring Boot; позже на этот навык мы будем опираться повсеместно;
- в главе 4 рассматриваются способы тестирования Spring-приложений, от самых простых компонентов до распределенных систем;
- в главе 5 описана незначительная реструктуризация, которую можно выполнить для перемещения приложения на такую облачную платформу, как Cloud Foundry, что позволит получить ряд ценных свойств от самой платформы;
- в главе 6 освещается вопрос создания HTTP- и RESTful-сервисов с помощью среды Spring; большей частью это будет делаться в API и в мире, ориентированном на предметную область;
- в главе 7 представлены общие способы контроля точек выдачи и поступления запросов в распределенных системах;
- в главе 8 описаны способы создания сервисов, действующих в качестве первого порта вызова для запросов, поступающих из внешнего мира;

- в главе 9 рассматриваются способы управления данными в среде Spring с помощью Spring Data, тем самым закладываются основы для мышления с прицелом на ориентацию в предметной области;
- в главе 10 изучаются методы интеграции распределенных сервисов и данных с использованием имеющейся в среде Spring архитектуры, управляемой событиями и ориентированной на рассылку сообщений;
- в главе 11 описаны способы применения возможностей масштабирования такой облачной платформы, как Cloud Foundry, для того, чтобы справиться с долговременными рабочими нагрузками;
- в главе 12 освещаются некоторые способы управления состоянием в распределенных системах;
- в главе 13 рассматриваются методы создания систем, поддерживающих отслеживаемость и оперативное реагирование;
- в главе 14 изучаются способы создания сервис-брокеров для платформ, подобных Cloud Foundry; сервис-брокеры подключают к облачной платформе такие сохраняющие состояние сервисы, как очереди сообщений, базы данных и блоки кэш-памяти;
- в главе 15 описаны творческие идеи, заложенные в непрерывную поставку. Это, может быть, и последняя глава, но для вас это всего лишь начало вашего путешествия.

Если вы похожи на нас, то не станете читать книгу от корки до корки. Если вы действительно похожи на нас, то, скорее всего, вообще не станете читать введение. Однако предположим, что вы видите эти строки. На такой случай хотим дать упреждающие советы.

- Чем бы вы ни занимались, прочтите главы 1 и 2. Они заложат основы для усвоения всего остального материала.
- В главах с 3-й по 6-ю вводятся такие понятия, которые должен знать любой проектировщик, работающий со средой Spring. Они актуальны как для прежних, так и для новых Spring-приложений. Фактически глава 5 посвящена вопросам превращения старых приложений (как ориентированных на использование среды Spring, так и не ориентированных на это) в новые приложения.
- В главах 7 и 8 вводятся понятия, применяющиеся в микросервисных системах, основанных на использовании протокола HTTP, например безопасность и маршрутизация.
- В главах с 9-й по 12-ю дается материал, помогающий лучше справляться с управлением и обработкой данных в распределенных системах.
- В главе 13 фактически изложена основная концепция, которая должна была быть освещена в этой книге значительно раньше, при рассмотрении ключевых концепций и тестирования, если не принимать во внимание тот факт, что она

зависит от технических концепций, которые ранее не вводились. Эксплуатируемые приложения должны быть отслеживаемыми. Эту главу следует прочитать как можно раньше. Для ее понимания достаточно знания основ. Или же перечитайте ее еще раз, добравшись до нее обычным порядком.

- В главе 14 рассматривается вопрос использования Spring, среды разработки облачных приложений, с целью создания компонентов для платформ, для облака. Материал, описанный в данной главе, прежде всего приобретает остроту в свете усилий открытых сервис-брокеров.
- И наконец, в главе 15 уточняются вопросы, связанные с непрерывной поставкой. Вся эта книга написана в понятиях непрерывной поставки, и последний раздел имеет такую фундаментальную значимость для всего, что мы стараемся сделать, что мы вынесли рассмотрение данного вопроса в заключение. Прочитайте эту главу одной из первых, а затем перечитайте ее в последнюю очередь.

Интернет-ресурсы

В Интернете имеется множество превосходных ресурсов, которые помогут продолжить ваши исследования и будут содействовать в работе:

- в GitHub-репозитории (https://github.com/cloud-native-java/) можно найти код для данной книги;
- сайт Spring (https://spring.io/) может послужить для вас универсальным магазином для всего, что касается Spring, включая документацию, форум технических вопросов и ответов и др.;
- сайт Cloud Foundry (https://www.cloudfoundry.org/) — центр притяжения той работы, которая проделана всеми соавторами учреждения Cloud Foundry. Здесь вы найдете видеоклипы, учебники, сводки новостей и др.

Условные обозначения

В этой книге используются следующие условные обозначения.

Курсив

Курсивом выделены новые термины и слова, на которых сделан акцент.

`Моноширинный шрифт`

Применяется для листингов программ, а также внутри абзацев, чтобы обратиться к элементам программы вроде переменных, функций и типов данных. Им также выделены имена и расширения файлов.

`Полужирный моноширинный шрифт`

Показывает команды или иной текст, который пользователь должен ввести самостоятельно.

Курсивный моноширинный шрифт

Показывает текст, который должен быть заменен значениями, введенными пользователем, или значениями, определяемыми контекстом.

Шрифт без засечек

Служит для указания URL, адресов электронной почты.

Этот рисунок указывает на совет или предложение.

Этот рисунок указывает на общее замечание.

Этот рисунок указывает на предупреждение.

Использование примеров кода

Загрузить дополнительные материалы (примеры кода, упражнения и т. д.) можно с сайта http://github.com/Cloud-Native-Java.

Книга нацелена на оказание помощи в решении практических задач. В целом, если пример кода предлагается вместе с книгой, то его можно использовать в ваших программах и документации. Вам не нужно обращаться к нам за разрешением, кроме тех случаев, когда вы воспроизводите весьма существенный объем кода. Например, написание программы, задействующей несколько фрагментов кода из этой книги, не требует разрешения, в отличие от продажи или распространения компакт-диска с примерами из книг издательства O'Reilly. Ответы на вопросы с цитированием материала данной книги и приведением примеров кода разрешения не требуют, чего не скажешь о вставке существенного объема примеров кода из этой книги в документацию по вашему программному продукту.

Если есть сомнения по поводу правомерности использования примеров кода в рамках вышеизложенного, вы можете свободно обратиться к нам за консультацией по адресу permissions@oreilly.com.

Благодарности

Прежде всего хочется поблагодарить наших невероятно терпеливых и участливых редакторов из O'Reilly: Нана Барбера (Nan Barber) и Брайана Фостера (Brian Foster).

Спасибо всем рецензентам и тем, кто нас вдохновлял, снабжал аналитическими выводами и идеями. Хочется выразить компании Pivotal и в целом всей ее экосистеме огромную признательность за поддержку. Особая благодарность каждому рецензенту, предоставившему технические отзывы, среди которых Брайан Дюссо (Brian Dussault), доктор Дэйв Сиер (Dave Syer), Эндрю Клей Шафер (Andrew Clay Shafer), Роб Уинч (Rob Winch), Марк Фишер (Mark Fisher), доктор Марк Поллак (Mark Pollack), Уткарш Надкарни (Utkarsh Nadkarni), Сабби Анандан (Sabby Anandan), Майкл Хангер (Michael Hunger), Бриджит Кромхаут (Bridget Kromhout), Расс Майлз (Russ Miles), Марк Хеклер (Mark Heckler), Мартен Дейнум (Marten Deinum), Натаниэль Шутта (Nathaniel Schutta), Патрик Крокер (Patrick Crocker) и многие другие. Спасибо вам!

И последними в списке, но не последними по значению хотелось бы поблагодарить Рода Джонсона и Джеймса Уоттерса. Вы проявили щедрость, не будучи ничем нам обязанными; вы дали нам все. Огромное вам спасибо за ваши предисловия, отзывы и напутствия.

Эта книга была написана с помощью инструментария Asciidoctor, разработанного под руководством Дэна Аллена (Dan Allen) (https://twitter.com/mojavelinux) и Сары Уайт (Sarah White) (http://twitter.com/carbonfray) из OpenDevise. Управление исходным кодом всех примеров выполняется в открытых хранилищах GitHub (https://github.com/). Код постоянно интегрировался в Travis CI (https://travis-ci.org/cloud-native-java). Компоненты сборок хранились и управлялись в личной учетной записи хранилища Artifactory, любезно предоставленной нам JFrog (https://jfrog.com/). Все

примеры запускались на Pivotal Web Services, где располагался экземпляр Cloud Foundry, управляемый Pivotal. Эти инструменты, без которых написание данной книги было бы сильно затруднено, относятся либо к программам с открытым кодом, либо запускаются сообществом единомышленников, не вынуждая нас ни к каким затратам. Огромное вам спасибо! Надеемся, что вы их присмотрите для своих следующих проектов и поддержите их, как они поддержали нас.

Теперь обращаемся к команде Spring (https://spring.io/team) и командам Cloud Foundry (https://www.cloudfoundry.org/): большое спасибо за весь созданный и протестированный вами код, которым нам не пришлось заниматься.

Джош Лонг

Хочу поблагодарить моего соавтора Кенни за то, что стал моим попутчиком в этом путешествии! Хочу выразить благодарность коллективу издательства O'Reilly за возможность работать с ними и за их невероятное терпение в условиях поджимающих сроков и нашего карабканья вверх в стремлении удержаться на вершине постоянно разрастающейся экосистемы Pivotal. Спасибо доктору Дэйву Сиеру (Dave Syer) (https://twitter.com/david_syer) за добрые слова о книге и за то, что он был моим вдохновителем. Меня носило по городам, часовым поясам, странам и континентам, а команды Spring и Cloud Foundry в Pivotal неизменно проявляли поддержку, независимо от дня или часа, — спасибо вам!

Кенни Бастани

В первую очередь хочу поблагодарить моего друга, наставника и коллегу Майкла Хангера (Michael Hunger), он был первым, кто привил мне страсть к написанию и обсуждению программ с открытым кодом. Хочу сказать спасибо и моему безмерно талантливому соавтору Джошу Лонгу (Josh Long), пригласившему меня поработать с ним над этой книгой, в то время как я запустил в эксплуатацию свои первые микросервисы Spring Boot более двух лет назад. Совместная работа над книгой была для меня невероятным приключением. С того времени как нашими совместными с Джошем усилиями на бумагу легли первые слова, мы увидели, что среда Spring Boot разрослась и стала одним из наиболее успешных когда-либо созданных проектов с открытым кодом. Сегодня рост ее популярности представлен почти 13 миллионами скачиваний в месяц, сочетанием новых приложений и непрерывного потока производственных развертываний. Достижением этого успеха команда Spring Engineering обязана 50 штатным инженерам. При этом скромном количестве специалистов Spring поддерживает более 100 проектов с открытым кодом, в числе которых компоненты среды, образцы приложений

и документация, и все это под крылом любезного спонсора — компании Pivotal. Все, что достигнуто нами в данной книге, не было бы возможным без невероятной самоотверженности разработчиков, которые в настоящее время ежегодно создают около трех миллионов новых приложений Spring Boot, готовых к работе в производственной среде.

С 2004 по 2017 год только в рамках проекта Spring Framework 381 Java-разработчик открытого кода внес 36 412 зафиксированных изменений, что составляет примерно 339 лет объединенных трудозатрат.

Часть I
Основы

1 Приложение, оптимизированное для работы в облачной среде

Приемы разработки программного обеспечения как в составе команд, так и в индивидуальном порядке постоянно совершенствуются. Разработка программ с открытым кодом обеспечила производство ПО чем-то похожим на Кембрийский взрыв инструментов, сред программирования, платформ и операционных систем, и все это сопровождается растущим вниманием к гибкости и автоматизации. Основная масса современного наиболее популярного инструментария с открытым кодом сфокусирована на функциональных особенностях, позволяющих командам разработчиков безостановочно поставлять программное обеспечение как никогда ранее быстрыми темпами, на любом уровне, от создания до практического применения.

История компании Amazon

В течение двух десятилетий, с начала 1990-х годов, книжный интернет-магазин Amazon.com со штаб-квартирой в Сиэтле превратился в крупнейшее в мире предприятие данного профиля. Теперь эта компания, известная в наши дни просто как *Amazon*, продает гораздо более широкий ассортимент товаров. В 2015 году Amazon превзошла по объему торговли компанию Walmart — наиболее заметного розничного торговца в США. Самая интересная часть истории беспрецедентного роста Amazon может быть сведена к одному вопросу: каким образом сайт, начинавшийся с простого книжного интернет-магазина, превратился в одного из мировых гигантов розничной торговли, не открыв при этом ни одной торговой точки?

Нетрудно было заметить, что мир торговли переместился и выстроился вокруг цифровых коммуникаций, чему поспособствовал бурный рост возможностей подключения к Интернету в любых уголках планеты. По мере уменьшения размеров персональных компьютеров и их превращения в повсеместно используемые в наши дни смартфоны, планшеты и часы мы стали свидетелями грандиозного роста доступности каналов распространения, преобразивших способы ведения торговли по всему миру.

Техническим развитием компании Amazon от весьма успешного книжного интернет-магазина до одной из наиболее влиятельных технологических компаний в мире, занимающихся розничной торговлей, управлял ее технический директор Вернер Фогельс (Werner Vogels). В июне 2006 года Фогельс дал интервью компьютерному журналу *ACM Queue* по вопросу быстрой эволюции технологий, которая привела к взлету компании Amazon. В нем директор рассказал об основном движущем факторе этого прогресса.

> В основном техническое развитие Amazon.com обуславливалось обеспечением непрерывного роста, **чтобы при исключительно высокой масштабируемости сохранялись доступность и производительность**.
>
> *Вернер Фогельс. A Conversation with Werner Vogels, ACM Queue 4, № 4 (2006): 14–22*

Фогельс продолжает утверждать, что компании Amazon для достижения высокой масштабируемости понадобился переход к другой структуре архитектуры ПО. Он напомнил, что сначала Amazon.com использовала монолитное приложение. Со временем, по мере того как над одним и тем же приложением стало трудиться все больше и больше команд, границы принадлежности кода стали размываться. «Отсутствие изолированности привело к отсутствию четко выраженной принадлежности», — сказал Фогельс.

Фогельс отметил, что совместно используемые ресурсы, такие как базы данных Vogels, затрудняют расширение всего бизнеса. Чем больше объем совместно применяемых ресурсов, будь то серверы приложений или базы данных, тем слабее контроль за работой при вводе компонентов программ в эксплуатацию.

> Вы создаете этот продукт, вам и обслуживать его работу.
>
> *Вернер Фогельс, технический директор компании Amazon*

Фогельс затронул общую тему, имеющую отношение и к приложениям, оптимизированным для работы в облачной среде: команды должны быть владельцами того,

что они создают. Далее он сказал: «Традиционная модель заключается в следующем: ПО перетаскивается через барьер, отделяющий создание от практического применения, работа над ним прекращается, после чего о нем забывают. Но это не про Amazon. *Вы создаете этот продукт, вам и обслуживать его работу*».

Слова, которые чаще всего цитировались основными докладчиками на отдельных ведущих конференциях по программному обеспечению *(«вы создаете этот продукт, вам и обслуживать его работу»)*, позже стали слоганом популярного движения, известного сегодня просто как *DevOps*.

Ряд приемов, о которых говорил Фогельс в 2006 году, дал начало многим популярным движениям в области разработки ПО, ныне процветающим. Можно установить связь таких подходов, как *DevOps* и *разработка микросервисов*, с идеями, высказанными Фогельсом более десяти лет назад. Идеи, аналогичные этим, вырабатывались в крупных интернет-компаниях, подобных Amazon, однако на создание соответствующих инструментов, позволяющих добиться их развития и становления в виде предложения конкретных услуг, ушло бы немало лет.

В 2006 году в Amazon был запущен новый продукт под названием Amazon Web Services (AWS). Идея, положенная в его основу, заключалась в предоставлении платформы, аналогичной той, что использовалась внутри Amazon, и выпуске ее в виде сервиса для открытого применения. Компания была заинтересована в развитии идей и инструментов, на которых основывалась эта платформа. Многие из выдвинутых Фогельсом идей уже были внедрены в платформе Amazon.com; выпуская платформу в качестве сервиса для открытого использования, компания стремилась выйти на новый рынок, названный *публичным облаком*.

Были озвучены идеи, положенные в основу этого облака. Виртуальные ресурсы могут предоставляться по требованию, при этом исключается необходимость беспокоиться относительно основной инфраструктуры. Разработчики могут просто арендовать виртуальную машину для размещения своих приложений, не нуждаясь в приобретении инфраструктуры или в управлении ею. Такой прием представлял собой вариант самообслуживания с низкой степенью риска, повышающий привлекательность публичного облака с технологией AWS, прокладывающей путь к внедрению.

Пройдут годы, прежде чем технология AWS превратится в полноценный набор сервисов и шаблонов для создания и запуска приложений, написанных для работы в публичном облаке. Хотя для создания новых программ к инжинирингу этих сервисов было привлечено множество разработчиков, многие компании с уже существующими приложениями по-прежнему испытывают сложности с переходом на новые технологии. Имеющиеся у них программы были созданы без расчета на переносимость. К тому же многие приложения все еще зависят от устаревших разработок, несовместимых с публичным облаком.

Чтобы самые крупные компании воспользовались публичным облаком, им нужно внести изменения в способ разработки их приложений.

Надежды, связанные с платформой

Платформа воспринимается сегодня как избитое понятие.

Когда речь заходит о платформах в компьютерном деле, чаще всего имеется в виду набор возможностей, помогающих либо создавать, либо запускать приложения. Лучше всего платформы обобщаются по требованиям, которые предъявляются к способам, используемым разработчиками при создании приложений.

Платформы позволяют автоматизировать решение задач, не играющих существенной роли в поддержке бизнес-требований, предъявляемых к приложению. Это помогает командам разработчиков действовать оперативно, позволяя поддерживать только функциональные компоненты, имеющие особую ценность для делового применения.

Та команда, которая написала сценарии оболочки либо наштамповала контейнеры или виртуальные машины для автоматизации разработки, уже создала своеобразную платформу. Вопрос заключается в следующем: что может дать эта платформа, каким ожиданиям соответствует? Какой объем работы потребуется для поддержки основных (или даже всех) требований для непрерывной поставки нового ПО?

При построении платформ создается инструмент, автоматизирующий набор повторяющихся приемов; последние складываются из набора требований, которые превращают ценные идеи в план.

- **Идеи:** каковы основные замыслы платформы и в чем их ценность?
- **Требования:** каковы требования, соблюдение которых необходимо для превращения наших идей в приемы?
- **Приемы:** как автоматически перевести требования в набор повторяющихся приемов?

В основу любой платформы закладываются простые идеи, которые в случае их реализации повышают дифференцированную деловую ценность за счет использования автоматизированного инструментария.

Возьмем, к примеру, платформу Amazon.com. Вернер Фогельс заявил: «Повышая изолированность компонентов программ друг от друга, команды будут иметь больше возможностей контроля тех функций, которые вводятся ими в производство».

Идея

Повышение изолированности компонентов программ друг от друга позволяет доводить части системы до практического применения независимо и в более сжатые сроки.

Закладывая эту идею в основу платформы, мы получаем возможность сформировать из нее набор требований. Они принимают форму взгляда на то, как основная идея будет воплощена на практике при автоматизации. Следующие утверждения

представляют собой категоричные требования к способам повышения изолированности компонентов.

Требования

- *Компоненты программ должны разрабатываться в виде независимо развертываемых сервисов.*
- *Вся бизнес-логика в сервисе инкапсулируется с теми данными, с которыми она работает.*
- *За пределами сервиса не должно быть прямого доступа к базе данных.*
- *Сервисы должны предоставлять веб-интерфейс для обращения к их бизнес-логике со стороны других сервисов.*

На основе этих требований складывается весьма категоричный взгляд на приемы усиления изолированности компонентов программы. Обещания, связанные с выполнением приведенных требований в виде автоматизированных механизмов, сулят командам предоставление более действенного контроля над функциями, поставляемыми для эксплуатации. Следующим шагом станет описание того, как эти требования могут быть сведены к набору повторяющихся приемов.

Приемы, выведенные исходя из этих требований, должны быть заявлены в виде коллекции обещаний. Указывая приемы в виде обещаний, мы задаем ожидания пользователей платформы относительно способов, которые они могут задействовать при создании и эксплуатации своих приложений.

Приемы

- *Командам предоставляется интерфейс самообслуживания, позволяющий обеспечить инфраструктуру, требуемую для функционирования приложений.*
- *Приложения помещаются в пакет, представляющий собой полный комплект, развертываются в среде с помощью интерфейса самообслуживания.*
- *Базы данных предоставляются приложениям в виде сервиса и должны вводиться в действие с использованием интерфейса самообслуживания.*
- *Приложение снабжается всем необходимым для регистрации в базе данных в виде переменных среды, но только после объявления явного отношения к базе данных в виде привязки к сервису.*
- *Каждому приложению предоставляется сервисный реестр, применяемый в качестве манифеста местоположений внешних сервисных зависимостей.*

Каждый из вышеперечисленных приемов принимает форму обещания для пользователя. Таким образом, суть идей, закладываемых в основу платформы, реализуется в виде требований, предъявляемых к приложениям.

В основе приложений, оптимизированных для работы в облачной среде, лежит набор требований, позволяющих сократить время, затрачиваемое на однообразную трудоемкую деятельность.

Когда технология AWS была впервые представлена публике, Amazon не заставляла ее пользователей придерживаться тех же требований, которые применялись в самой компании. Оставаясь верным своему названию *Amazon Web Services*, сама по себе технология AWS не является облачной *платформой*, а представляет собой коллекцию независимых инфраструктурных сервисов, из которых может быть составлена автоматическая оснастка, напоминающая платформу обещаний. Спустя годы после первого выпуска AWS компания Amazon начала предлагать коллекцию управляемых сервисов платформы с вариантами использования, начиная от Интернета вещей (IoT, Internet of Things) и заканчивая машинным самообучением.

Если какой-либо компании понадобится создать свою собственную платформу с нуля, время поставки приложений откладывается до полной сборки платформы. Компании, ставшие ранними пользователями AWS, должны были вместе собрать некую разновидность автоматизации, похожую на платформу. Каждой компании пришлось бы выработать набор обещаний, охватывающих основные идеи создания и ввода ПО в эксплуатацию.

Совсем недавно производство ПО ориентировалось на идею существования основного набора общих обещаний, исполняемых каждой облачной платформой. Эти обещания будут исследоваться в нашей книге с применением *платформы в виде сервиса* (Platform as a Service, PaaS) под названием Cloud Foundry. Основным предназначением последней является предоставление платформы, вбирающей в себя набор общих обещаний, касающихся быстрой разработки и применения приложений. Cloud Foundry дает эти обещания, одновременно сохраняя возможность переносимости между облачными инфраструктурами нескольких различных поставщиков.

Основное внимание в данной книге мы уделим вопросам создания Java-приложений, оптимизированных для работы в облачной среде. В первую очередь это будет касаться инструментов и сред, позволяющих сократить время, затрачиваемое на однообразные трудоемкие действия за счет использования преимуществ и обещаний платформы, оптимизированной для работы в облаке.

Принципы

Новые принципы создания ПО позволяют уделять больше внимания обдумыванию поведения наших приложений в ходе их применения. В совместных усилиях разработчиков и операторов делается более серьезный акцент на понимании характера поведения их приложений в процессе эксплуатации, с меньшим уровнем доверия к тому, как за счет сложности можно разрешить ситуацию при сбое.

Как и в случае с Amazon.com, архитектура ПО начала отходить от крупных монолитных приложений. Теперь основное внимание в вопросах архитектуры уделялось достижению высокого уровня масштабируемости без ущерба для производительности и доступности. Разбивая монолит на компоненты, инженерные организации

предпринимали усилия по децентрализации управления изменениями, предоставляя командам больше контроля над тем, как функции вводятся в эксплуатацию. Повышая изолированность между компонентами, команды создателей ПО начали вступать в мир разработки распределенных систем, фокусируясь на написании менее крупных, более специализированных сервисов с независимыми циклами выпуска.

В приложениях, оптимизированных для выполнения в облаке, используется набор принципов, позволяющих командам более свободно оперировать способами ввода функций в эксплуатацию. По мере роста распределенности приложений (в результате повышения степени изолированности, необходимой для предоставления большего контроля над ситуацией командам, владеющим приложением) возникает серьезная проблема, связанная с повышением вероятности сбоя при обмене данными между компонентами приложения. Неизбежным результатом превращения приложений в сложные распределенные системы становятся эксплуатационные сбои.

Архитектуры приложений, оптимизированных для работы в облачной среде, придают этим приложениям преимущества исключительно высокой масштабируемости, притом гарантируя их всеобщую доступность и высокий уровень производительности. Хотя такие компании, как Amazon, пользовались преимуществами высокой масштабируемости в облаке, широко доступного инструментария для создания приложений, оптимизированных для работы в облачной среде, еще не появлялось. В конечном итоге инструменты и платформа будут представлены в виде коллекции проектов с открытым кодом, поддерживаемых первопроходцем в мире открытых облачных вычислений — компанией Netflix.

Масштабируемость

Чтобы ускорить разработку ПО, нужно заранее продумывать возможности масштабирования на всех уровнях. В самом общем смысле *масштаб* обуславливается издержками, приносящими пользу. Уровень непредсказуемости, сокращающий этот полезный эффект, называется *риском*. В таких условиях приходится определять границы масштабирования, поскольку создание программ сопряжено с рисками. Риски, заложенные создателями ПО, не всегда известны операторам, то есть тем, кто занимается его эксплуатацией. Выдвигая разработчикам требования по скорейшему вводу функций в дело, мы добавляем риски к процессу эксплуатации этих функций, не испытывая никакого сочувствия к проблемам операторов.

В результате со стороны операторов возрастает недоверие к ПО, произведенному разработчиками. Дефицит доверия между обеими сторонами создает практику обвинений: люди предъявляют друг другу претензии вместо того, чтобы рассматривать системные проблемы, которые привели к возникновению или к ускорению появления проблем, оказывающих отрицательное влияние на ведение дел.