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.

Для снятия напряженности, возникающей в традиционных структурах ИТ-организаций, следует переосмыслить порядок взаимодействия команд, занимающихся поставкой и эксплуатацией ПО. Общение операторов с разработчиками способно влиять на возможность масштабирования, поскольку со временем цели каждой из сторон расходятся. Чтобы добиться успеха в решении данного вопроса, необходимо перейти к более надежному способу разработки ПО, при котором основное внимание внутри процесса создания уделяется опыту команды операторов и который способствует взаимообучению и совершенствованию.

Надежность

Ожидания, возникающие во взаимоотношениях команд (относительно порядка разработки, режима эксплуатации, алгоритма взаимодействия с пользователем и т. д.), выражаются в виде соглашений. Заключаемые командами соглашения подразумевают предоставление или получение определенного уровня услуг. Наблюдая за тем, как команды предоставляют услуги друг другу в процессе создания ПО, можно прийти к более глубокому пониманию того, каким образом отсутствие надлежащего уровня общения способно повлечь возникновение рисков, приводящих к сбоям при дальнейшей совместной работе.

Соглашения об услугах между командами заключаются с целью уменьшить риск неожиданного поведения в общих функциях масштабирования, которые создают ценность для бизнеса. Данные соглашения носят конкретный характер, чтобы гарантировать соответствие поведения ожидаемой стоимости операций. Таким образом, услуги позволяют отдельным частям бизнеса довести до максимальных показателей его общую отдачу. Здесь целью для бизнеса, относящегося к производству ПО, является надежное прогнозирование создания ценности за счет затрат, то есть то, что мы называем *надежностью*.

Модель услуг для бизнеса — та же самая модель, которая используется при создании ПО. Именно так гарантируется надежность системы, неважно, к чему именно это относится: к программному продукту, производимому для автоматизации бизнес-функции, или к людям, обучаемым выполнению ручных операций.

Адаптивность

Мы начинаем понимать, что больше не существует лишь одного способа разработки и практического применения программного обеспечения. Благодаря внедрению адаптивных методологий и продвижению в направлении к бизнес-моделям ПО как услуги (или сервиса) — *Software as a Service* (SaaS) — комплект корпоративных приложений становится все более распределенным. Создание распределенных систем — весьма непростая задача. Переход к более распределенной архитектуре

приложений обуславливается необходимостью сокращать сроки поставки программ с меньшим риском возникновения сбоев.

Возможно, кто-то воскликнет: «Адаптивность? Она все еще актуальна?» (https://www.linkedin.com/pulse/agile-dead-matthew-kern). Адаптивность в нашем понимании относится как к целостному, общесистемному устремлению к незамедлительной поставке новой ценности, так и к замыслу по поводу быстрого реагирования на смену обстановки. Мы просто говорим о малом — об адаптивности, не касаясь при этом понятий управленческой практики. Существует множество путей, приводящих к эксплуатации продукта, и нас не интересует, какой именно методики управления вы будете придерживаться на своем пути. Главное, вам нужно усвоить, что адаптивность — полезное свойство, а не самоцель.

Современный бизнес, ориентированный на производство ПО, стремится реструктурировать процессы разработки, чтобы получить возможность более быстро внедрять программы и непрерывно вводить приложения в эксплуатацию. Компании хотят увеличить не только скорость разработки программ, но и количество создаваемых и сопровождаемых ими приложений, чтобы обслуживать различные бизнес-подразделения организации.

ПО все чаще становится для компаний конкурентным преимуществом. Все более совершенные инструменты позволяют бизнес-специалистам открывать новые источники дохода или оптимизировать бизнес-функции таким образом, чтобы можно было обеспечить быстрое внедрение инноваций.

И в центре всего этого движения находится *облако*. Когда о нем заходит речь, подразумевается весьма специфичный набор технологий, позволяющий разработчикам и операторам сопровождения пользоваться преимуществами веб-сервисов, существующих для предоставления виртуализированной вычислительной инфраструктуры и управления ею.

Компании начинают переходить от дата-центров к применению публичных облаков. Одной из таких является Netflix, популярная компания по предоставлению потокового мультимедиа по подписке.

История Netflix

На сегодняшний день Netflix — один из самых крупных в мире потоковых медиасервисов по требованию, эксплуатирующий свои онлайн-сервисы в облаке. Компания была основана в 1997 году в Скоттс-Валли, штат Калифорния, Ридом Хастингсом (Reed Hastings) и Марком Рэндольфом (Marc Randolph). Изначально Netflix предоставляла интернет-сервис по прокату DVD, позволявший клиентам вносить фиксированную ежемесячную плату за подписку на не-

ограниченные прокатные видео без дополнительной платы. Клиенты получали DVD по почте после выбора их изображения из списка и помещения в очередь с помощью сайта Netflix.

В 2008 году компания пережила серьезное повреждение своей базы данных, помешавшее доставке DVD клиентам. В те времена Netflix уже приступила к развертыванию сервисов потокового видео, предназначенных для обслуживания клиентов. Команда, занимавшаяся в Netflix потоковым видео, поняла, что аналогичный отказ в их сфере услуг подорвет будущее их бизнеса. В результате в компании было принято важное решение: нужно переходить к другому способу разработки и сопровождения ПО, гарантирующему постоянную доступность их сервисов клиентам.

Частью решения Netflix по предотвращению сбоев ее интернет-сервисов стал отход от вертикально масштабируемой архитектуры и единых точек отказов. Такая позиция была вызвана повреждением базы данных в результате использования вертикально масштабируемой реляционной базы данных. Компания переместила данные клиентов в распределенную базу данных NoSQL, а именно в проект Apache Cassandra с открытым кодом. Это было началом становления Netflix в качестве компании «облачного направления», запускающей все свои программные приложения в виде отказоустойчивых облачных сервисов с высокой степенью распределения. Netflix решила повысить надежность своих интернет-сервисов, добавляя избыточность к своим приложениям и базам данных в масштабируемой модели инфраструктуры.

Частично решение компании по переходу в облако потребовало переноса развертывания больших приложений на системы с высокой степенью распределенности. При этом специалисты компании столкнулись с серьезной проблемой: командам Netflix при переходе от собственного дата-центра к использованию публичного облака пришлось заниматься перепроектированием своих приложений. В 2009 году компания приступила к переходу на Amazon Web Services (AWS) и сконцентрировалась на достижении трех основных целей: масштабируемости, производительности и доступности.

В начале 2009 года стало ясно, что спрос резко возрастает. Известен такой факт: Юрий Израилевский (Yury Izrailevsky), вице-президент по проектированию облачных вычислений и платформ (Cloud and Platform Engineering) компании Netflix, на презентации в рамках конференции AWS re:Invent, состоявшейся в 2013 году, сказал, что спрос с 2009 года возрос в 100 раз. «Мы не смогли бы справиться с масштабированием наших сервисов, если бы пользовались своим внутренним решением», — сказал Израилевский.

Кроме того, он заявил, что преимущества масштабируемости в облаке на фоне его быстрого глобального расширения стали еще более очевидными. Израилевский сказал: «Чтобы сократить время ожидания для наших европейских клиентов, мы запустили второй облачный район в Ирландии. На развертывание нового

дата-центра на другой территории ушло бы много месяцев и миллионы долларов. Это были бы громадные вложения».

Как только Netflix начала размещать свои приложения на Amazon Web Services, сотрудники стали делиться впечатлениями в блоге компании. Многие поддерживали переход к новой архитектуре, сконцентрированной на горизонтальной масштабируемости на всех уровнях стека ПО.

Джон Цианкутти (John Ciancutti), занимавший тогда должность вице-президента технологий персонализации (Personalization Technologies) компании Netflix, в конце 2010 года выложил в блог сообщение, что «облачная среда идеально подошла для горизонтально масштабируемых архитектур. Нам не нужно гадать на месяцы вперед, какими будут наши потребности в оборудовании, хранилище и сетевой аппаратуре. Мы можем практически мгновенно программным способом получить доступ к большему объему этих ресурсов из общих пулов в Amazon Web Services».

Под возможностью «программного доступа» к ресурсам Цианкутти подразумевал, что разработчики и операторы сопровождения могут получить программный доступ к конкретным API управления, открываемым Amazon Web Services с целью предоставить клиентам средства управления для подготовки виртуализированной вычислительной инфраструктуры. RESTful API позволяют разработчикам создавать приложения, управляющие виртуальной инфраструктурой и подготавливающие ее для их основных приложений.

Слоеный пирог, показанный на рис. 1.1, изображает параметры облака, характеризуемые различными уровнями абстракции.

Предоставление управленческих услуг для управления виртуализированной вычислительной инфраструктурой — одна из основных концепций облачных вычислений и называется инфраструктурой как услугой (Infrastructure as a Service), чаще всего обозначаемой IaaS.

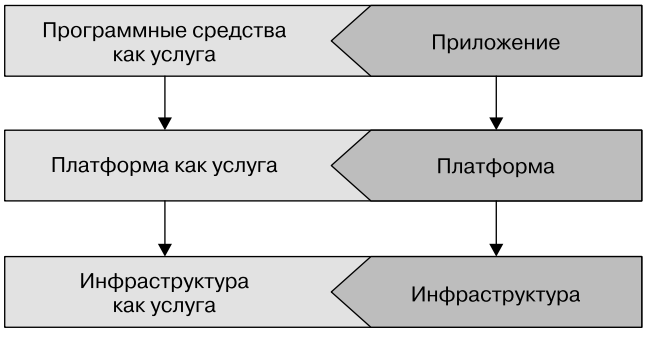

Рис. 1.1. Стек облачных вычислений