ГЛАВА 1

Обзор разработки корпоративных приложений

Разработка корпоративных приложений всегда была одной из самых интересных областей разработки программного обеспечения, а последнее десятилетие оказалось особенно захватывающим периодом. В 2010-х годах высокораспределенные микросервисы постепенно вытеснили классические трехуровневые архитектуры, а почти безграничные ресурсы облачной инфраструктуры обусловили устаревание тяжеловесных серверов приложений. Столкнувшись со сложностями объединения частей распределенных архитектур, многие сомневаются в том, что этот сложный мир микросервисов необходим. Реальность такова, что большинство приложений все еще остаются хорошо продуманными монолитами, которые развиваются в соответствии с традиционными принципами разработки программного обеспечения.

Однако способы развертывания и эксплуатации программного обеспечения изменились очень быстро. Мы наблюдаем, как DevOps перерастает в GitOps, расширяя обязанности разработчиков за пределы кода приложения и включая необходимую инфраструктуру. Эта книга, основанная на книге *Modern Java EE Design Patterns* Маркуса Эйзеле (Markus Eisele) (O'Reilly; https://oreil.ly/1cROz), рассматривает модернизацию в более широ-

ком смысле, чем просто модульная организация программ. Мы хотим помочь вам разобраться в различных компонентах современной платформы разработки на основе Kubernetes и понять, как создавать и поддерживать приложения на ее основе.

Цель книги — дать возможность оценить факторы успеха и движущие силы модернизации приложений и облачных архитектур. Все свое внимание мы уделим модернизации корпоративных приложений на Java, включая процесс выбора приложений, пригодных для модернизации, и обзор инструментов и методологий, помогающих управлять модернизацией. Вместо того чтобы обсуждать паттерны, мы приведем примеры, которые помогут применить ваши знания.

Мы не будем противопоставлять монолитные и распределенные приложения, а постараемся помочь вам понять, как перенести ваши приложения в облако, обойдясь минимумом затрат.

Вы можете использовать книгу как справочник и читать главы в любом порядке. Однако мы организовали материал так, что повествование начинается с описания концепций высокого уровня и заканчивается реализацией. Кроме того, мы считаем важным сначала познакомиться с различными понятиями облачных сред, а затем начинать создавать для них приложения.

От общего к частному. В чем привлекательность облаков

Различия между общедоступными, частными и гибридными облаками и многооблачными средами (мультиоблаками) когда-то легко определялись местоположением и владением. Сегодня два этих признака больше не являются единственными важными факторами классификации облаков. Начнем с более полного определения различных сред и выясним, чем они хороши.

Общедоступная облачная среда обычно создается из ресурсов, не принадлежащих конечному пользователю, и они могут перераспределяться между арендаторами. Ресурсы частной облачной среды принадлежат исключительно конечному пользователю и обычно находятся за пользовательским брандмауэром в центре обработки данных или (иногда) на локальной машине. Совокупность нескольких облачных сред с определенной степенью переносимости, оркестрации и управления рабочими нагрузками называется гибридным облаком. А совокупность несвязанных и независимых облаков обычно называют мультиоблаком. Гибридный и мультиоблачный подходы являются взаимоисключающими — нельзя иметь и то и другое одновременно, поскольку облака будут либо взаимосвязаны (гибридное облако), либо нет (мультиоблако).

Корпоративные приложения все чаще развертываются в облаках разных типов, так как предприятия стремятся повысить безопасность и производительность за счет расширенного портфеля сред. Но безопасность и производительность — лишь две из многих причин переноса рабочих нагрузок в гибридные или мультиоблачные среды. Основной мотивацией для многих людей является модель «плати только за то, что используешь». Вместо того чтобы вкладывать средства в дорогостоящее локальное оборудование, которое сложно и дорого масштабировать, можно использовать ресурсы облака и только в необходимом объеме. При этом вам не нужно вкладывать деньги в оборудование, коммунальные услуги или строительство собственного центра обработки данных. Вам даже не нужны специальные IT-команды для управления вашим облачным центром обработки данных, поскольку вы можете воспользоваться опытом сотрудников облачного провайдера.

Для разработчиков облако подразумевает самообслуживание и гибкость, избавляя от необходимости ждать развертывания среды и позволяя выбирать необходимые инфраструктурные компоненты (например, базы данных, брокеры сообщений и т. д.), что в конечном счете ускоряет цикл разработки. Помимо этих основных преимуществ, можно воспользоваться специальными функциями, доступными разработчикам в некоторых облачных средах. Так, OpenShift имеет встроенную консоль разработки,

позволяющую напрямую редактировать детали топологии приложений. Облачные IDE (например, Eclipse Che (https://www.eclipse.org/che)) предоставляют доступ к рабочим областям разработки через браузер и устраняют необходимость настраивать локальную среду для команд.

Кроме того, облачные инфраструктуры поддерживают автоматизацию процессов развертывания, позволяющую развертывать программное обеспечение в тестовых и промышленных средах одним нажатием кнопки, — обязательное требование, предъявляемое группами, использующими методики гибкой разработки и DevOps. Те, кто знаком с разработкой архитектур микросервисов, не понаслышке знают о необходимости 100%-ной автоматизации. Но она выходит далеко за рамки прикладных частей: распространяется на инфраструктуру и нижестоящие системы. В этом вам помогут Ansible (https://www.ansible.com), Helm (https://helm.sh) и операторы Kubernetes (https://oreil.ly/lhaPm). Подробнее об автоматизации мы поговорим в главе 4, а тему операторов Kubernetes затронем в главе 7.

Что означает «облачный»

Вы, наверное, слышали об *облачном* подходе к разработке приложений и сервисов, особенно после того, как в 2015 году была основана организация Cloud Native Computing Foundation (CNCF), выпустившая Kubernetes v1. Билл Уайлдер (Bill Wilder) впервые использовал термин «облачный» в своей книге *Cloud Architecture Patterns* (O'Reilly; https://oreil.ly/hmeAC). По словам Уайлдера, облачное приложение спроектировано так, чтобы в полной мере использовать преимущества облачных платформ за счет применения сервисов этих платформ и автоматического масштабирования. Уайлдер написал свою книгу в период растущего интереса к разработке и развертыванию облачных приложений. Разработчики могли выбирать из широкого круга различных общедоступных и частных платформ, включая Amazon AWS, Google Cloud, Microsoft Azure, и множества других платформ менее известных провайдеров. Но в то же время все более распространенным становилось развертывание гибридных облаков, что создавало проблемы.

Вот как CNCF (https://oreil.ly/Sadph) определяет термин «облачный»:

> Облачные технологии позволяют организациям создавать и запускать масштабируемые приложения в современных динамичных средах — в общедоступных, частных и гибридных облаках. Примерами таких технологий могут служить контейнеры, сервисные сетки, микросервисы, неизменяемая инфраструктура и декларативные API.
>
> Эти технологии обеспечивают устойчивость, управляемость и наблюдаемость слабосвязанных систем. В сочетании с надежной автоматизацией они позволяют инженерам часто вносить важные изменения, дающие предсказуемый результат, с минимальными трудозатратами.
>
> *CNCF Cloud Native Definition v1.0*

Подобными облачными технологиями являются двенадцатифакторные приложения (https://12factor.net/ru/). Соответствующий манифест определяет паттерны для создания приложений, доставляемых через облако. Эти паттерны пересекаются с паттернами облачной архитектуры Уайлдера, но двенадцатифакторная методология может применяться к приложениям, которые написаны на любом языке программирования и используют любую комбинацию вспомогательных сервисов (баз данных, очередей, кэш-памяти и т. д.).

Разработка для Kubernetes

Для разработчиков, развертывающих приложения в гибридном облаке, имеет смысл сместить акцент с понятия «облачный» на понятие «для Kubernetes». Одно из первых упоминаний «для Kubernetes» имело место еще в 2017 году. В статье в блоге Medium описываются различия между понятиями «для Kubernetes» (https://oreil.ly/2quU8) и «облачный» как обусловленные набором технологий, оптимизированных для Kubernetes. Основной вывод заключается в том, что к приложениям «для Kubernetes» относятся специализированные приложения, которые одновременно яв-

ляются облачными. Если просто облачные приложения предназначены для выполнения в облаке, то приложения для Kubernetes разрабатываются для выполнения под управлением Kubernetes.

На заре облачной разработки различия в оркестрации не позволяли приложениям быть по-настоящему облачными. Kubernetes решает проблему оркестрации, но не распространяется на сервисы облачных провайдеров (например, Roles и Permissions) и не предоставляет шину событий (например, Kafka). Идея о том, что приложения для Kubernetes являются специализированными облачными, означает, что между ними есть много общего. Основное различие заключается в независимости от облачного провайдера. Использование всех преимуществ гибридного облака и совместимость с разными облачными провайдерами требует, чтобы приложения можно было развернуть в облаке любого провайдера. Без этого вы будете привязаны к одному облачному провайдеру и вам придется полагаться на его безупречную работу. Чтобы в полной мере использовать преимущества гибридного облака, приложения должны создаваться на основе Kubernetes. Разработка для Kubernetes решает проблему переносимости. Подробнее об особенностях создания приложений для Kubernetes мы поговорим в главе 2.

Контейнеры и оркестрация для разработчиков

Один из ключевых элементов переносимости — *контейнер*. Он служит представлением доли ресурсов хост-системы вместе с приложением. Контейнеры появились еще на заре Linux по мере введения изолированных сред chroot и стали основным направлением развития контейнеров процессов Google, которые в конечном итоге превратились в контрольные группы cgroups. Их популярность резко возросла в 2013 году, в первую очередь благодаря появлению Docker, сделавшему их доступными для многих разработчиков. Важно различать Docker как компанию, контейнеры Docker, образы Docker и инструменты разработчика Docker, к которым мы все привыкли. Все началось с контейнеров Docker, однако Kubernetes позволяет запускать контейнеры в любой среде выполнения

контейнеров (например, containerd (https://containerd.io) или CRI-O (https://cri-o.io)), поддерживающей интерфейс среды выполнения контейнера (Container Runtime Interface, CRI). То, что многие называют образами Docker, на самом деле является образами, упакованными в формате Open Container Initiative (OCI) (https://opencontainers.org/).

Среда выполнения контейнеров

Контейнеры предлагают облегченную версию пространства пользователя в операционной системе Linux, урезанную до самого необходимого. Тем не менее контейнер — это все еще операционная система, и качество контейнера имеет такое же значение, как и основная операционная система. Поддержка образов контейнеров требует многих затрат на проектирование, анализа безопасности и оценки ресурсов, необходимых для поддержки образов контейнеров. Как следствие, нужно тестировать не только сами образы, но и их поведение на заданном хосте. Использование сертифицированных и совместимых с OCI базовых образов устраняет препятствия, которые могут возникать при перемещении приложений между платформами. В идеале базовые образы уже поставляются с необходимыми средами выполнения для разных языков. Для приложений на основе Java хорошей основой является универсальный базовый образ Red Hat (https://oreil.ly/KH9od). Больше о контейнерах и о том, как их используют разработчики, мы поговорим в главе 4.

Разновидности Kubernetes

До сих пор мы обсуждали Kubernetes как общую концепцию. И мы продолжим использовать слово *Kubernetes*, говоря о технологии, лежащей в основе оркестрации контейнеров. Название Kubernetes (иногда просто K8s) относится к проекту с открытым исходным кодом (https://kubernetes.io), широко известному как свод стандартов для основных функций оркестрации контейнеров. В книге мы будем использовать «простой» термин Kubernetes, рассуждая о стандартной функциональности внутри Kubernetes. Сообщество Kubernetes создало множество

разных дистрибутивов и даже разновидностей Kubernetes. Организация CNCF запустила сертифицированную программу соответствия Kubernetes (Certified Kubernetes Conformance Program (https://oreil.ly/n4XH9)), в которой на момент написания этих слов было перечислено более 138 продуктов от 108 производителей. Список включает полные дистрибутивы (например, MicroK8s, OpenShift, Rancher), предложения хостинга (например, Google Kubernetes Engine, Amazon Elastic Kubernetes Service, Azure AKS Engine) и инсталляторы (например, minikube, VanillaStack). Все они имеют общее ядро, но добавляют дополнительные функции или средства, которые, по мнению производителей, будут востребованы. В этой книге мы не делаем никаких предложений о том, какой вариант Kubernetes использовать. Вам придется самостоятельно решить, с помощью чего вы будете управлять своими рабочими нагрузками. Но, чтобы помочь вам локально опробовать примеры из книги, мы предлагаем использовать minikube (https://oreil.ly/sCQUo) и не просим выполнять полноценную установку где-то в облаке.

Управление сложностью разработки

Один из наиболее важных аспектов разработки для Kubernetes — управление средой разработки. Количество задач, которые необходимо решить для успешного развертывания в нескольких средах, увеличивается в геометрической прогрессии. Одна из причин — растущее количество отдельных частей приложений или микросервисов. Еще одна причина — конфигурация инфраструктуры для конкретного приложения. На рис. 1.1 представлен краткий обзор типичной среды с инструментами, необходимыми для полностью автоматизированной разработки. Мы поговорим о некоторых из них в книге, чтобы помочь вам начать работать в новой среде. Основные задачи разработки не изменились. Вы по-прежнему будете писать приложения или сервисы, используя подходящие фреймворки, такие как Quarkus (https://quarkus.io), как это делаем мы в книге. Данная часть рабочего процесса разработки обычно называется внутренним циклом разработки.