
Оглавление

Предисловие	13
Введение	15
Условные обозначения	16
Использование примеров кода	17
Платформа онлайн-обучения O'Reilly	18
Благодарности	18
ГЛАВА 1. Путь к эксплуатации	21
Что такое Kubernetes	21
Основные компоненты	22
Не только оркестрация: дополнительные функции	24
Интерфейсы Kubernetes	24
Kubernetes в целом	26
Что такое платформа приложений	27
Спектр подходов	28
Спектр подходов с учетом потребностей вашей организации	29
Платформы приложений: подводим итоги	31
Создание платформ приложений на основе Kubernetes	31
Начиная снизу	33
Спектр абстрагирования	34
Определение возможностей платформы	36
Составные компоненты	37
Резюме	41
ГЛАВА 2. Модели развертывания	42
Управляемые сервисы и самостоятельное развертывание	42
Управляемые сервисы	43
Самостоятельное развертывание	43
Принятие решения	44
Автоматизация	45
Готовый установщик	45
Собственные средства автоматизации	46

Архитектура и топология	47
Модели развертывания etcd	47
Уровни кластера	49
Пулы узлов	50
Федерация кластеров	52
Инфраструктура	55
Физическое и виртуальное оборудование	56
Выбор размера для кластера	59
Вычислительная инфраструктура	61
Сетевая инфраструктура	62
Стратегии автоматизации	64
Развертывание серверов	66
Управление конфигурацией	66
Системные образы	67
Что устанавливать	67
Контейнерные компоненты	69
Дополнения	70
Обновления	72
Версионирование платформы	73
Планирование на случай сбоев	73
Интеграционное тестирование	74
Стратегии	75
Механизмы инициирования	81
Резюме	82
ГЛАВА 3. Среда выполнения контейнеров	83
Появление контейнеров	83
Open Container Initiative	85
Спецификация ОСИ для сред выполнения	85
Спецификация ОСИ для образов	87
Интерфейс среды выполнения контейнеров	90
Запуск Pod'a	90
Выбор среды выполнения	92
Docker	93
containerd	94
CRI-O	95
Kata Containers	96
Virtual Kubelet	98
Резюме	98
ГЛАВА 4. Хранилище данных контейнера	100
Требования к хранилищу	100
Режимы доступа	101
Расширение томов	101
Выделение томов	102

Резервное копирование и восстановление.....	102
Блочные устройства и хранение файлов/объектов	103
Временные данные	103
Выбор провайдера хранилища.....	104
Механизмы для работы с хранилищами в Kubernetes	104
Постоянные тома и заявки на выделение	104
Классы хранилищ.....	107
CSI	108
Контроллер CSI.....	109
Узел CSI.....	110
Реализация хранилища в виде сервиса.....	110
Установка компонентов CSI	110
Предоставление разных вариантов хранилищ	113
Использование хранилища.....	115
Изменение размера	117
Копии (snapshots)	118
Резюме.....	120
ГЛАВА 5. Сетевое взаимодействие между Pod'ами	121
Аспекты, связанные с сетью	122
Управление IP-адресами	122
Протоколы маршрутизации	124
Инкапсуляция и туннелирование	126
Маршрутизируемость приложений.....	127
IPv4 и IPv6	128
Шифрование трафика рабочих заданий.....	128
Сетевая политика	129
Аспекты, связанные с сетью: итоги	131
Интерфейс управления сетью контейнеров (CNI)	132
Установка CNI.....	133
Подключаемые модули CNI.....	136
Calico	136
Cilium	140
AWS VPC CNI	143
Multus	144
Дополнительные подключаемые модули	145
Резюме.....	146
ГЛАВА 6. Маршрутизация сервисов	147
Сервисы Kubernetes.....	148
Компонент Service	148
Endpoints	154
Аспекты реализации Сервиса	158
Обнаружение сервисов	168
Производительность DNS-сервиса.....	170

Ingress	172
Зачем нужен механизм Ingress	172
API-интерфейс Ingress	173
Контроллеры Ingress и принцип их работы	176
Методы маршрутизации входящего трафика	177
Выбор контроллера Ingress	181
Вопросы, связанные с развертыванием контроллера Ingress	183
DNS-сервер и его роль в обработке входящего трафика	185
Управление сертификатами TLS	187
Mesh-сеть	189
Где (не) следует использовать mesh-сети	190
Интерфейс mesh-сети	191
Прокси-сервер плоскости данных	194
Mesh-сеть в Kubernetes	196
Архитектура плоскости данных	201
Внедрение mesh-сети	202
Резюме	206
ГЛАВА 7. Управление конфиденциальными данными	207
Углубленная защита	208
Шифрование дисков	209
Безопасность во время передачи	210
Прикладное шифрование	211
Secret API в Kubernetes	211
Модели потребления объектов <i>Secret</i>	213
Конфиденциальные данные в <i>etcd</i>	216
Шифрование с использованием статического ключа	218
Шифрование методом конвертов	222
Внешние провайдеры	224
Vault	224
Cyberark	225
Интеграция путем внедрения	225
Интеграция CSI	230
Конфиденциальные данные в декларативном мире	232
Запечатывание конфиденциальных данных	233
Обновление ключей	236
Многокластерные модели	237
Рекомендации по работе с конфиденциальными данными	237
Всегда проводите аудит взаимодействия с конфиденциальными данными	238
Не раскрывайте конфиденциальные данные	238
Отдавайте предпочтение томам перед переменными окружения	238
Делайте так, чтобы ваши приложения не знали о провайдерах хранилищ для конфиденциальных данных	239
Резюме	239

ГЛАВА 8. Управление допуском.....	240
Цепочка допуска в Kubernetes	241
Встроенные контроллеры допуска	242
Веб-хуки.....	243
Настройка контроллеров допуска на основе веб-хуков	245
Аспекты проектирования веб-хуков	247
Написание изменяющего веб-хука	248
Простой HTTPS-обработчик.....	249
Controller Runtime	251
Системы с централизованными политиками.....	254
Резюме.....	261
ГЛАВА 9. Наблюдаемость	262
Принцип работы журналирования	262
Обработка журнальных записей контейнера	263
Журналы аудита в Kubernetes.....	266
События Kubernetes	268
Генерация оповещений на основе журнальных записей.....	270
Последствия для безопасности.....	270
Метрики	270
Prometheus.....	271
Долгосрочное хранение	272
Пассивная модель сбора метрик.....	272
Пользовательские метрики	273
Организация метрик и федеративные системы.....	274
Оповещения	275
Потребляемые ресурсы и их стоимость.....	276
Компоненты для работы с метриками	280
Распределенная трассировка.....	288
OpenTracing и OpenTelemetry	289
Компоненты трассировки.....	290
Инструментирование приложений.....	291
Mesh-сети.....	291
Резюме.....	292
ГЛАВА 10. Идентификация	293
Идентификация пользователей.....	294
Методы аутентификации.....	295
Выдача пользователям минимальных привилегий.....	306
Идентификация контейнерных приложений.....	309
Общие секреты.....	310
Сетевая идентификация.....	311
Токены служебной учетной записи.....	315

Прогнозируемые токены служебной учетной записи	318
Идентификация узлов на уровне платформ.....	321
Резюме.....	333
ГЛАВА 11. Создание сервисов платформы	335
Механизмы расширения	336
Подключаемые расширения.....	336
Расширения на основе веб-хуков	337
Операторы	338
Шаблон проектирования "оператор".....	339
Контролеры Kubernetes	339
Пользовательские ресурсы.....	341
Сценарии использования операторов	344
Служебные компоненты платформы	345
Операторы приложений общего назначения.....	346
Операторы для отдельно взятых приложений	346
Разработка операторов.....	347
Инструментарий для разработки операторов.....	347
Проектирование моделей данных	351
Реализация бизнес-логики.....	353
Расширение планировщика	370
Предикаты и приоритеты	370
Политики планирования.....	371
Профили планирования.....	372
Несколько планировщиков	373
Создание собственного планировщика.....	373
Резюме.....	373
ГЛАВА 12. Мультиотенантность	374
Уровни изоляции.....	374
Однотенантные кластеры.....	375
Мультиотенантные кластеры.....	376
Разделение на основе пространств имен.....	377
Мультиотенантность в Kubernetes.....	379
Управление доступом на основе ролей.....	379
Квоты на ресурсы.....	381
Веб-хуки допуска	382
Запросы и лимиты на ресурсы	384
Сетевые политики	389
Политики безопасности Pod	392
Сервисы мультиотенантных платформ.....	395
Резюме.....	397

ГЛАВА 13. Автоматическое масштабирование	398
Виды масштабирования.....	399
Архитектура приложения.....	400
Автомасштабирование приложений	401
Horizontal Pod Autoscaler	401
Vertical Pod Autoscaler	405
Автомасштабирование с помощью пользовательских метрик.....	408
cluster-proportional-autoscaler	409
Создание собственных средств автомасштабирования.....	410
Автомасштабирование кластера.....	410
Выделение резервных ресурсов для кластера.....	414
Резюме.....	416
ГЛАВА 14. Эффективная эксплуатация приложений	417
Развертывание приложений в Kubernetes	418
Шаблонизация манифестов развертывания	418
Упаковка приложений для Kubernetes	419
Получение конфигурации и конфиденциальных данных	419
Объекты <i>ConfigMap</i> и <i>Secret</i>	420
Получение конфигурации из внешних систем.....	423
Реакция на события перепланирования	424
Хуки, срабатывающие перед остановкой контейнеров.....	424
Безопасное завершение работы контейнеров.....	425
Удовлетворение требований доступности.....	427
Проверки состояния	429
Проверки работоспособности	429
Проверки готовности.....	430
Проверки состояния запуска.....	431
Реализация проверок	432
Запросы и лимиты на ресурсы Pod.....	432
Запросы ресурсов	433
Лимиты на ресурсы.....	434
Журналирование приложений	434
Что записывать в журнал	435
Структурированные и неструктурированные журнальные записи.....	435
Контекстная информация в журнальных записях	436
Предоставление метрик.....	436
Инструментирование приложений.....	436
Метод USE.....	438
Метод RED	438
Четыре "золотых" сигнала	439
Метрики отдельных приложений.....	439

Инструментирование сервисов для распределенной трассировки.....	439
Инициализация трассировщика.....	440
Создание спанов.....	441
Передача контекста.....	442
Резюме.....	443
ГЛАВА 15. Логистика доставки программного обеспечения	444
Создание образов контейнеров.....	445
Плохая практика "золотых" базовых образов	447
Выбор базового образа	448
Выбор пользователя для выполнения контейнера.....	449
Явное определение версий пакетов.....	450
Образы для сборки и выполнения приложений.....	450
Cloud Native Buildpacks	451
Реестры образов	453
Сканирование уязвимостей.....	454
Процедура карантина	456
Подписание образов	457
Непрерывная доставка.....	458
Интеграция процесса сборки в конвейер.....	459
Развертывание на основе загрузки	462
Методы выкатывания изменений	464
GitOps.....	466
Резюме.....	467
ГЛАВА 16. Абстрагирование платформы.....	469
Открытость платформы.....	469
Самостоятельное присоединение к платформе.....	471
Спектр абстрагирования.....	473
Инструменты командной строки.....	474
Абстрагирование посредством шаблонизации	475
Абстрагирование стандартных компонентов Kubernetes.....	479
Полностью скрываем Kubernetes.....	482
Резюме.....	485
Об авторах	487
Об изображении на обложке	488
Предметный указатель	489

Предисловие

С тех пор как мы сделали платформу Kubernetes общедоступной, прошло уже больше шести лет. Я участвовал в данном проекте с самого начала и, между прочим, зафиксировал в его репозитории первую порцию кода (все выглядело не настолько впечатляюще, как может показаться; это был всего лишь формальный этап создания нового репозитория для публичного выпуска). Могу с уверенностью сказать, что успех Kubernetes оказался для нас довольно неожиданным; он стал возможным благодаря тесному сотрудничеству преданных и доброжелательных участников проекта, а также помощи тех, кто использует Kubernetes на практике.

Мне посчастливилось поработать с авторами этой книги в стартапе (Neptio), который был основан (в том числе и мною) для того, чтобы способствовать внедрению Kubernetes в типичные предприятия. Причиной успеха Neptio во многом стали усилия моих коллег по налаживанию прямой связи с теми пользователями Kubernetes, которые решают реальные проблемы. Я благодарен каждому из них. Данная книга вобрала в себя этот опыт, полученный из первых рук. Она предоставляет командам разработчиков инструменты, необходимые для успешного применения Kubernetes.

Моя профессиональная карьера с самого начала была ориентирована на построение систем, предназначенных для тех, кто разрабатывает приложения, как в одиночку, так и в составе команды. Все началось с Microsoft Internet Explorer и продолжилось работой над Windows Presentation Foundation, после чего я переключился на облачные технологии, такие как Google Compute Engine и Kubernetes. Снова и снова я наблюдал за тем, как создатели платформ сталкивались со своего рода проклятием. Те, кто занимается разработкой платформ, мыслят более долгосрочными категориями и сосредоточены на создании фундамента, который, как они надеются, прослужит не одно десятилетие. Но из-за этого они теряют из виду проблемы, с которыми пользователи сталкиваются прямо сейчас. Нередко мы настолько увлекаемся процессом разработки, что у нас не остается времени на оценку того, что мы создали.

Снять это проклятие можно только путем активного сбора информации за пределами тесного сообщества создателей платформ. Для меня это стало возможным благодаря работе в команде инженерно-технического обеспечения Neptio (и позже в группе проектирования Kubernetes [Kubernetes Architecture Team или KAT] в VMware). В результате удалось успешно внедрить Kubernetes разнообразным клиентам в ряде отраслей, а также выяснить, как на самом деле применяется "теория", заложенная в нашу платформу.

Данная проблема усугубляется еще и потому, что вокруг Kubernetes и CNCF (Cloud Native Computing Foundation) сформировалась активно развивающаяся "экосистема" из проектов, входящих в фонд CNCF, и тех, которые не имеют к нему прямого

отношения. Я люблю называть подобную экосистему "прекрасным хаосом". Это джунгли, в которых произрастают проекты разной степени зрелости, иногда с одинаковыми возможностями. Именно так выглядят инновации! Однако исследование экосистемы, как и джунглей, является рискованным и требует самоотверженности и настойчивости. Новичкам в мире Kubernetes зачастую не хватает времени или умений для того, чтобы как следует овладеть экосистемой этой платформы.

В данной книге описываются те части экосистемы, в которых уместно применять отдельные инструменты и средства, а также демонстрируется, как подобрать правильное решение частных проблем, с которыми сталкивается читатель. Но мы не просто советуем читателю выбрать тот или иной инструмент. Наш подход более общий, он требует понимания сути проблемы, решаемой с помощью определенной категории инструментов, уверенности в наличии этой конкретной проблемы, осведомленности о преимуществах и недостатках разных методов, и предлагает практические рекомендации относительно того, с чего начать. Это бесценная информация для тех, кто собирается применять Kubernetes в реальных условиях!

В завершение хотел бы сказать большое спасибо Джошу, Ричу, Алексу и Джону. Их опыт имеет прямое отношение к успеху многих клиентов. Они научили меня многому при работе над проектом, который мы основали более шести лет назад, и теперь множество других пользователей могут воспользоваться нашим уникальным опытом, прочитав эту книгу.

Джо Беда,
главный инженер VMware Tanzu,
соавтор Kubernetes,
Сиэтл, январь 2021 года.

Введение

Kubernetes — необычайно мощная технология, получившая стремительный взлет популярности. Она подготовила почву для реального прогресса в сфере управления процессом развертывания программного обеспечения. В момент появления Kubernetes ПО на основе API и распределенные системы, может, и не были широко распространены, но уже хорошо себя зарекомендовали. Проект Kubernetes стал отличным воплощением этих принципов, что послужило залогом его успеха. Но вместе с тем он привнес еще кое-что крайне важное. В недалеком прошлом ПО, способное самостоятельно достигать объявленного, желаемого состояния, существовало только в гигантских технологических компаниях, в которых работали группы самых талантливых инженеров. Теперь же высокодоступные, самовосстанавливающиеся, автоматически масштабирующиеся средства развертывания доступны любой организации, и все благодаря Kubernetes. Перед нами открывается будущее, в котором программные системы принимают от нас высокоуровневые команды общего характера и, действуя в соответствии с ними, обеспечивают нужные результаты за счет анализа сложившихся условий, преодоления постоянно меняющихся преград и решения проблем без нашего вмешательства. Более того, эти системы будут справляться со своими задачами быстрее и надежнее, чем люди, работающие вручную. Благодаря Kubernetes мы с вами приблизились к этому будущему. Однако за мощь и богатые возможности приходится платить повышенной сложностью. Решение написать эту книгу было мотивировано нашим желанием поделиться своим опытом и помочь другим совладать со сложными аспектами данной платформы.

Если вы хотите использовать Kubernetes для создания приложений промышленного уровня, эта книга для вас. Если же вам нужно познакомиться с системой Kubernetes или понять, как она работает, то лучше поискать какую-то другую книгу. Эти темы раскрыты во множестве других изданий, в официальной документации, в бесчисленных статьях и в самом исходном коде. Советуем вам подкрепить чтение этой книги самостоятельным исследованием и тестированием обсуждаемых здесь решений, так как мы редко предоставляем примеры в виде подробного пошагового руководства. Мы пытаемся охватить столько теоретического материала, сколько необходимо, а процесс реализации в основном оставляем читателю в качестве упражнения.

На страницах этой книги вы найдете рекомендации о параметрах, инструментарии, шаблонах проектирования и их практическом применении. Читатель должен понимать, как авторы относятся к практике создания платформ приложений. Мы инженеры и архитекторы; мы сотрудничаем со многими компаниями из рейтинга Fortune 500, помогая им воплотить представления о том, как должна выглядеть платформа в производственном окружении. Kubernetes помогает нам в этом аж с

2015 года, когда вышла версия 1.0. Мы сделали все возможное для того, чтобы сосредоточиться на проектировании и философии, а не на инструментарии, поскольку новые инструменты появляются настолько быстро, что мы не успеваем о них писать. Тем не менее, мы обязаны продемонстрировать эти шаблоны проектирования с применением самых подходящих на сегодняшний день инструментов.

Нам удалось добиться значительных успехов, помогая группам разработчиков на их пути к переходу в облако и полной трансформации процессов создания и развертывания своего программного обеспечения. Но не обошлось и без неудач, частой причиной которых было непонимание руководства организации того, какие задачи решает Kubernetes. Вот почему мы с самого начала удаляем этому пристальное внимание. За это время мы обнаружили несколько областей, которые в особенности интересуют наших клиентов. Совещания, направленные на то, чтобы помочь нашим клиентам продвинуться на пути к развертыванию версии или даже просто определить этот путь, стали рутинными. Нам приходилось проводить их настолько часто, что мы решили написать книгу!

Мы снова и снова помогали организациям в прохождении этого пути к развертыванию их ПО, но во всех этих случаях просматривалась лишь одна закономерность: данный процесс каждый раз выглядит иначе, хотя иногда нам очень хотелось, чтобы это было не так. Мы хотим, чтобы вы отдавали себе отчет в том, что в настоящей книге не стоит искать "программы из пяти шагов" по развертыванию кода в производственном окружении или "10 вещей, которые должен знать каждый пользователь Kubernetes". Здесь мы будем обсуждать многочисленные моменты принятия решений и ловушки, с которыми мы сталкивались, при необходимости мы будем приводить конкретные примеры и случаи из жизни. Общепринятые рекомендации существуют, но к ним всегда следует относиться прагматично. Нет какого-то одного универсального подхода, а фраза "зависит от обстоятельств" является совершенно справедливым ответом на многочисленные вопросы, с которыми вы неизбежно столкнетесь на своем пути.

Тем не менее, мы призываем вас относиться к этой книге критически! Работая с клиентами, мы всегда приветствуем критику и предложения с их стороны. Знания необходимо постоянно оттачивать, поэтому мы периодически обновляем наши методики с учетом новых возможностей, информации и ограничений. Вы должны следовать нашему примеру. Поскольку сфера облачных технологий продолжает эволюционировать, вы, несомненно, будете отклоняться от наших рекомендаций. Мы лишь делимся здесь своим опытом, чтобы вы могли сопоставить нашу точку зрения со своей собственной.

Условные обозначения

В этой книге приняты следующие типографские обозначения:

◆ *Курсивный шрифт*

Служит для выделения новых терминов, URL-адресов, адресов электронной почты, имен и расширений файлов.

◆ Моноширинный шрифт

Предназначен для записи листингов программ, а также для выделения в тексте таких элементов, как имена переменных и функций, базы данных, типы данных, переменные окружения, операторы и ключевые слова.

◆ Полужирный моноширинный шрифт

Используется для выделения команд или другого текста, который пользователь должен вводить без каких-либо изменений.

◆ Курсивный моноширинный шрифт

Обозначает те элементы в коде, которые требуется заменить предоставленными пользователем значениями или значениями, зависящими от контекста.

Термины, относящиеся к Kubernetes, начинаются с большой буквы: Pod-оболочка*, Сервис и StatefulSet.

Чтобы акцентировать внимание читателя, в некоторых местах в текст вставлены соответствующие значки:



Данный элемент обозначает совет или предложение.



Этот значок обозначает примечание общего характера.



Данный элемент обозначает предупреждение или предостережение.

Использование примеров кода

Дополнительный материал (примеры программного кода, упражнения и т. д.) доступен для загрузки по адресу <https://github.com/production-kubernetes>.

Если при использовании примеров кода у вас возникнут технические вопросы или проблемы, пожалуйста, обращайтесь к нам по адресу bookquestions@oreilly.com.

Цель этой книги — помочь в решении ваших задач. Если предлагается какой-то пример кода, его разрешается использовать в ваших программах и документации. Обращаться к нам, если вы не воспроизводите его существенную часть, не нужно, например, в ситуации, когда вы включаете в свою программу несколько фрагментов кода, приведенного здесь. Однако продажа или распространение CD-ROM с примерами из книг издательства O'Reilly требует отдельного разрешения. Цитировать эту книгу с примерами кода при ответе на вопрос вы можете свободно, но, если хотите включить существенную часть приведенного здесь кода в документацию своего продукта, вам следует связаться с нами.

* В тексте сохранен оригинальный английский термин Pod. — Прим. ред.

Мы приветствуем, но не требуем наличия ссылки на оригинал. Ссылка обычно состоит из названия, имени автора, издательства, ISBN и копирайта. Например, *Kubernetes на практике*, Джош Россо, Рич Ландер, Александр Бранд и Джон Харрис (O'Reilly). Copyright 2021 Josh Rosso, Rich Lander, Alexander Brand, and John Harris, 978-1-492-09231-5.

Если вы сомневаетесь в том, что работа с примерами кода не выходит за рамки добросовестного использования или не нарушает условий, перечисленных выше, то можете обратиться к нам по адресу permissions@oreilly.com.

Платформа онлайн-обучения O'Reilly

На протяжении более 40 лет издательство O'Reilly Media (<http://oreilly.com>) проводит технические и бизнес-тренинги, делится знаниями и опытом, чтобы помочь компаниям достичь успеха.

Наше уникальное сообщество экспертов и новаторов делится знаниями и опытом с помощью книг, статей, конференций и нашей платформы онлайн-обучения. Эта платформа предоставляет доступ к учебным курсам, проводимым в прямом эфире, углубленные учебные программы, интерактивные среды для написания кода и обширную коллекцию текстов и видео от O'Reilly и более чем 200 других издательств. Больше информации можно найти на странице <http://oreilly.com>.

Благодарности

Авторы хотели бы поблагодарить Кэти Гаманжи, Майкла Гуднесса, Джима Уебера, Джеда Салазара, Тони Скалли, Монику Родригез, Криса Докери, Ральфа Бэнкстона, Стова Слоку, Аарона Миллера, Тунде Олу-Ису, Алекса Уитроу, Скотта Лоу, Райана Чаппла и Кенана Дервисевича за их отзывы о рукописи. Спасибо Полу Лундину за поддержку в написании этой книги и за создание замечательной команды инженерно-технического обеспечения в Nertio. Все члены этой команды внесли свой вклад, участвуя в развитии множества идей и практических подходов, которые рассматриваются на последующих страницах данной книги. Спасибо Джо Беде, Скотту Бьюкенену, Даниелле Барроу и Тиму Ковентри-Коксу из VMware за их поддержку в основании этого проекта и работе над ним. Наконец, мы благодарим Джона Девинса, Джеффа Блейела и Кристофера Фаучера из O'Reilly за их постоянную поддержку.

Авторы также хотели бы лично выразить признательность следующим людям:

Джош. Хотел бы поблагодарить Джессику Аппельбаум за ее невероятную поддержку, в частности за олады с голубикой, которые она мне готовила, пока я работал над этой книгой. Также хочу сказать спасибо моей маме, Анжеле, и папе, Джо, за то, что с детства были моей опорой.

Рич. Хотел бы поблагодарить мою жену, Тейлор, и детей, Райну, Жасмин, Макса и Джона, за их поддержку и понимание, которые они проявили, пока я работал над

этой книгой. Также хочу сказать спасибо моей маме, Дженни, и папе, Норму, за то, что были отличными примерами для подражания.

Александр. Шлю свои любовь и благодарность моей жене, Анаис, которая оказывала мне всестороннюю поддержку, пока я уделял время написанию этой книги. Также признателен моей семье, друзьям и коллегам, которые помогли мне стать тем, кем я являюсь сегодня.

Джон. Хотел бы поблагодарить мою прекрасную жену, Кристину, за ее любовь и терпение во время моей работы над этой книгой. Также говорю спасибо моим близким друзьям и семье за непрерывную помощь и поддержку на протяжении всех этих лет.

Издательство «БХВ» искренне благодарит за помощь в подготовке русскоязычного издания этой книги научного редактора и рецензента — руководителя группы архитекторов и системных инженеров Cros Code Дмитрия Бардина.

Путь к эксплуатации

https://t.me/it_books

За годы своего существования платформу Kubernetes внедрили многие организации. Росту ее популярности, бесспорно, способствует распространение контейнерных рабочих заданий и наличие микросервисов. Когда команды, занимающиеся эксплуатацией, обслуживанием инфраструктуры и разработкой, сталкиваются с необходимостью создания, выполнения и поддержки этих рабочих процессов, некоторые из них выбирают в качестве решения Kubernetes. Это довольно молодой проект в сравнении с другими продуктами с открытым исходным кодом, такими как Linux. Как могут подтвердить многие клиенты, с которыми мы сотрудничали, большинство пользователей Kubernetes только начинают работать с этой платформой. Несмотря на ее применение во многих организациях, она редко встречается в эксплуатации и еще реже — в высоконагруженных проектах. В этой главе мы подготовимся к пути, на котором находятся многие команды инженеров, работающие с Kubernetes. В частности, мы рассмотрим некоторые аспекты, на которые следует обращать внимание при планировании процесса подготовки проекта к эксплуатации.

Что такое Kubernetes

Что такое Kubernetes? Платформа? Инфраструктура? Приложение? На этот счет есть много мнений, приверженцы которых могут дать вам свое определение. Вместо того чтобы высказать еще одно мнение, мы сосредоточимся на классификации тех задач, которые решает Kubernetes. После этого мы поговорим о том, как, опираясь на этот набор возможностей, достигнуть результатов, соответствующих производственному уровню. Идеальный исход прочтения этой книги — ситуация, при которой рабочие задания успешно работают с реальным трафиком.

Название *Kubernetes* в какой-то степени носит общий характер. Если заглянуть на GitHub, можно заметить, что организация kubernetes содержит (на момент написания этих строк) 69 репозиторийев. А ведь есть еще и kubernetes-sigs со 107 репозиториями. Не говоря уже о сотнях проектов в составе фонда CNCF (Cloud Native Computing Foundation), существующих в этой среде! В данной книге под Kubernetes имеется в виду лишь основной проект с одноименным названием. Какой из них основной? Тот, который находится в репозитории kubernetes/kubernetes (<https://github.com/kubernetes/kubernetes>). Именно там хранятся все компоненты, которые встречаются в большинстве кластеров. От кластера, состоящего из этих компонентов, можно ожидать следующих возможностей:

- ◆ планирование распределения рабочей нагрузки для множества хостов;
- ◆ предоставление декларативного, расширяемого API-интерфейса для взаимодействия с системой;

- ◆ наличие утилиты командной строки `kubectl` для взаимодействия с API-сервером в ручном режиме;
- ◆ приведение объектов от текущего состояния к желаемому;
- ◆ предоставление базового служебного слоя для направления запросов к приложениям и обратно;
- ◆ наличие нескольких интерфейсов для поддержки подключаемых модулей, отвечающих за работу с сетью, хранилищем и т. д.

Перечисленные возможности делают данный проект, согласно утверждению его разработчиков, *системой оркестрации контейнеров, готовой к эксплуатации*. Говоря простым языком, Kubernetes дает нам возможность исполнять и распределять контейнеризированные приложения на разных хостах. Помните об этом по мере того, как мы будем углубляться в подробности. Надеемся, со временем нам удастся продемонстрировать, что эта возможность, несмотря на ее основополагающий характер, является лишь одним из этапов на пути к эксплуатации.

Основные компоненты

Какие компоненты обеспечивают функциональность, которая рассматривается в этой книге? Как уже упоминалось, основные компоненты находятся в репозитории `kubernetes/kubernetes`. Многие из нас применяют их по-разному. Например, те, кто пользуются управляемыми сервисами наподобие Google Kubernetes Engine (GKE), скорее всего, уже имеют все эти компоненты на своих хостах. Кто-то может скачивать исходный код из репозитория или получать подписанные версии от поставщика. Как бы то ни было, выпуски Kubernetes доступны для свободного скачивания в репозитории `kubernetes/kubernetes`. Распаковав такой выпуск, вы можете получить соответствующие двоичные файлы с помощью команды `cluster/get-kubebinaries.sh`. Она автоматически определит вашу целевую архитектуру и скачает серверные и клиентские компоненты. Давайте рассмотрим это на примере следующего кода и затем проанализируем ключевые компоненты:

```
$ ./cluster/get-kube-binaries.sh
```

```
Kubernetes release: v1.18.6
Server: linux/amd64 (to override, set KUBERNETES_SERVER_ARCH)
Client: linux/amd64 (autodetected)
```

```
Will download kubernetes-server-linux-amd64.tar.gz from https://dl.k8s.io/v1.18.6
Will download and extract kubernetes-client-linux-amd64.tar.gz
Is this ok? [Y]/n
```

Внутри архива со скачанными серверными компонентами, который, скорее всего, был сохранен в папку `server/kubernetes-server- $\{ARCH\}$.tar.gz`, находятся ключевые элементы, составляющие кластер Kubernetes:

- ◆ *API-сервер* — главное место взаимодействия для всех компонентов Kubernetes и пользователей. Именно здесь мы получаем, добавляем, удаляем и изменяем объекты. API-сервер делегирует работу с состоянием внутреннему компоненту, роль которого чаще всего играет `etcd`.

- ◆ *Kubelet* — агент, размещенный на хосте и взаимодействующий с API-сервером для получения информации о состоянии узла и определения того, выполнение каких приложений следует на нем планировать. В пределах хоста он взаимодействует со средой выполнения контейнеров, такой как Docker, обеспечивая корректные запуск и выполнение приложений, запланированных на соответствующем узле.
- ◆ *Диспетчер контроллеров* — группа обработчиков, которая объединена в одно приложение и занимается приведением многих важных объектов Kubernetes к желаемому состоянию. Когда такое состояние (например, "в Развертывании должно быть три реплики") объявляется, для его достижения внутренний контроллер берет на себя создание новых подов.
- ◆ *Планировщик* определяет, где должны выполняться приложения, в зависимости от того, какой узел он считает оптимальным. Для принятия этого решения он использует фильтрацию и систему оценок.
- ◆ *Kube-proxy* реализует сервисы Kubernetes, предоставляя виртуальные IP-адреса, способные направлять трафик к подам. Это достигается за счет механизма фильтрации пакетов на хосте, такого как iptables или ipvs.

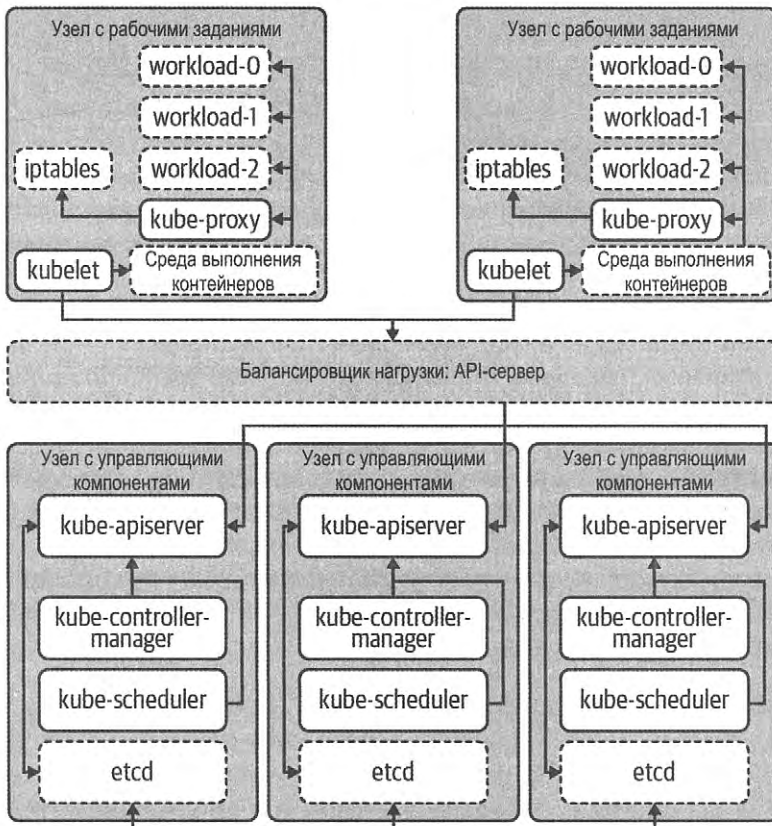


Рис. 1.1. Ключевые компоненты, составляющие кластер Kubernetes. Компоненты, обранные пунктирной линией, не входят в ядро Kubernetes

Несмотря на то, что это не исчерпывающий список, но в нем перечислены главные компоненты, составляющие основу той функциональности, которую мы обсудили. Если говорить в контексте архитектуры, то на рис. 1.1 показано, как взаимодействуют эти компоненты.



Архитектура Kubernetes имеет много вариантов. Например, во многих кластерах компоненты kube-apiserver, kube-scheduler и kube-controller-manager выполняются в виде контейнеров. Это означает, что мастер-узел или управляющий слой (control-plane) может также включать container-runtime, kubelet и kube-proxy. Подобные аспекты развертывания будут рассмотрены в следующей главе.

Не только оркестрация: дополнительные функции

В некоторых областях Kubernetes не ограничивается одной лишь оркестрацией приложений. Как уже упоминалось, компонент kube-proxy делает так, чтобы хосты автоматически предоставляли приложениям виртуальные IP-адреса, которые ведут к одной или нескольким внутренним подам. Очевидно, что это выходит за рамки выполнения и планирования контейнеризированных приложений. Вместо реализации всего этого в ядре Kubernetes теоретически можно было бы определить API-интерфейс для работы с сервисами, требуя, чтобы он был реализован в виде подключаемого модуля. В результате такого подхода мы были бы вынуждены выбирать между многочисленными модулями, доступными в экосистеме, вместо того чтобы задействовать функциональность, встроенную в ядро.

Это модель, которая применяется во многих API-интерфейсах Kubernetes, таких, как NetworkPolicy. Например, создание объекта Ingress в кластере Kubernetes не гарантирует выполнение действия. Иными словами, API-интерфейс хоть и существует, но находится за пределами ядра. Разработчики должны выбрать технологию, на основе которой этот API-интерфейс будет реализован. В случае с Ingress многие используют такой контроллер, как ingress-nginx (<https://kubernetes.github.io/ingress-nginx>), который работает в кластере. Он реализует API-интерфейс, считывая объекты Ingress и создавая конфигурационные файлы для экземпляров NGINX, которые указывают поды. Однако ingress-nginx — это лишь один из множества вариантов. Project Contour (<https://projectcontour.io>) реализует тот же Ingress API, но при этом конфигурирует экземпляры Envoy, прокси-сервера, лежащего в основе Contour. Благодаря наличию подключаемых модулей разработчикам есть из чего выбрать.

Интерфейсы Kubernetes

Развивая эту идею расширения функциональности, мы должны теперь рассмотреть интерфейсы. Интерфейсы Kubernetes позволяют нам изменять и дополнять основные возможности этой платформы. Мы относимся к интерфейсу как к определению или контракту о том, как должно происходить взаимодействие с тем или иным компонентом. В сфере разработки программного обеспечения это созвучно принципу определения функциональности, которую могут реализовывать классы или структуры. В таких системах, как Kubernetes, для удовлетворения требований этих

интерфейсов мы развертываем подключаемые модули, предоставляя возможности наподобие управления сетью.

Конкретным примером отношений между интерфейсами и подключаемыми модулями является CRI (Container Runtime Interface — интерфейс среды выполнения контейнеров; <https://github.com/kubernetes/cri-api>). На ранних этапах развития платформа Kubernetes поддерживала всего одну среду выполнения контейнеров, Docker. И хотя Docker по-прежнему присутствует во многих кластерах, наблюдается растущий интерес к альтернативам, таким как containerd (<https://containerd.io>) и CRI-O (<https://github.com/cri-o/cri-o>). На рис. 1.2 вышеупомянутые отношения проиллюстрированы на примере этих двух сред выполнения контейнеров.

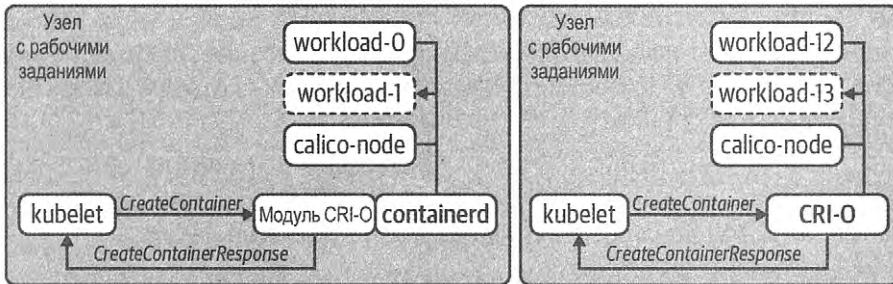


Рис. 1.2. Две разные среды выполнения контейнеров, установленных на двух узлах. Kubelet шлет такие запросы, как `CreateContainer`, определенные в CRI, в расчете на то, что среда выполнения их удовлетворит и вернет ответ

Во многих интерфейсах такие команды, как `CreateContainerRequest` или `PortForwardRequest` выполняются в виде удаленного вызова процедур (англ. Remote Procedure Calls или RPC). В случае с CRI взаимодействие происходит по gRPC, а kubelet ожидает получения ответов вроде `CreateContainerResponse` и `PortForwardResponse`. На рис. 1.2 можно также увидеть две разные модели удовлетворения требований CRI. Модуль CRI-O с самого начала создавался в качестве реализации CRI, поэтому kubelet передает ему эти команды напрямую. А вот containerd поддерживает подключаемый модуль, который служит прослойкой между kubelet и собственными интерфейсами. Какой бы ни была конкретная архитектура, главное, чтобы агенту kubelet не нужно было знать о том, как организована работа среды выполнения контейнеров в *каждом* отдельном случае. Именно этот принцип делает интерфейсы настолько мощным средством проектирования, разработки и развертывания кластеров Kubernetes.

Мы видели, как со временем часть функций была вынесена из основного проекта в соответствии с упомянутым принципом подключаемых модулей. Речь идет о вещах, которые существовали в кодовой базе `kubernetes/kubernetes` по историческим причинам. Примером этого являются средства интеграции `cloud-provider` (<https://github.com/kubernetes/cloud-provider>; CPI). Большинство модулей CPI традиционно встраивались в такие компоненты, как `kube-controller-manager` и `kubelet`. Они, как правило, отвечали за выделение балансировщиков нагрузки, предоставление доступа к метаданным облачных провайдеров и пр. Иногда, особенно

до создания CSI (Container Storage Interface — интерфейс хранилищ для контейнеров; <https://kubernetes-csi.github.io/docs/introduction.html>), эти провайдеры выделяли блочные хранилища и делали их доступными для приложений, выполняемых в Kubernetes. Это довольно обширная функциональность даже для ядра, не говоря уже о том, что ее нужно было заново реализовывать для каждого отдельно взятого провайдера! В качестве более разумного решения эти возможности были вынесены в отдельную модель интерфейса, `kubernetes/cloud-provider` (<https://github.com/kubernetes/cloud-provider>), реализацией которой могут заниматься разные проекты и поставщики. В результате удалось не только упорядочить кодовую базу Kubernetes, но и обеспечить возможность управления этой функциональностью вне основных кластеров Kubernetes. Сказанное относится и к таким распространенным процедурам, как обновление и устранение уязвимостей.

На сегодня существует несколько интерфейсов, которые позволяют видоизменять и дополнять возможности Kubernetes. Вот общий список, который мы расширим в следующих главах этой книги:

- ◆ CNI (Container Networking Interface — интерфейс управления сетью контейнеров) позволяет сетевым провайдерам определять, как они выполняют свои функции, от IPAM до непосредственной маршрутизации пакетов.
- ◆ CSI (Container Storage Interface — интерфейс хранилищ для контейнеров) позволяет провайдерам хранилищ удовлетворять запросы рабочих заданий в пределах кластера. Обычно реализуется для таких технологий, как vSAN и EBS.
- ◆ CRI (Container Runtime Interface — интерфейс среды выполнения контейнеров) обеспечивает поддержку различных сред выполнения, в том числе таких популярных, как Docker, containerd и CRI-O. Также способствовал распространению менее традиционных сред выполнения, включая firecracker, которая использует KVM для создания минимальных виртуальных машин.
- ◆ SMI (Service Mesh Interface — интерфейс mesh-сети сервисов) — это один из новых интерфейсов, появившихся в экосистеме Kubernetes. Он создавался в надежде унифицировать определение таких понятий, как "политика маршрутизации трафика", "телеметрия" и "управление".
- ◆ CPI (Cloud Provider Interface — интерфейс облачных провайдеров) позволяет провайдерам, таким как VMware, AWS, Azure и др., создавать для своих облачных сервисов точки интеграции с кластерами Kubernetes.
- ◆ Спецификация среды выполнения OCI (Open Container Initiative) стандартизирует форматы образов так, чтобы образы контейнеров, созданные в соответствии с ней, могли выполняться в любой среде, совместимой с OCI. Она не имеет прямого отношения к Kubernetes, но поддерживает стремление к реализации подключаемых сред выполнения контейнеров (CRI).

Kubernetes в целом

Теперь давайте поговорим о сфере применения Kubernetes. Это *средство оркестрации контейнеров* с некоторыми дополнительными функциями. Его можно рас-

ширять и изменять за счет подключаемых модулей для интерфейсов. Kubernetes может стать основополагающим инструментом для организаций, которым нужен изящный способ выполнения своих приложений. Но давайте сделаем шаг назад. Если взять системы для выполнения приложений, которые используются в вашей организации в настоящий момент, и заменить их платформой Kubernetes, будет ли этого достаточно? Во многих случаях компоненты и механизмы, составляющие "платформу приложений", имеют множество нюансов.

Нам неоднократно приходилось наблюдать за муками, вызванными необдуманном выбором так называемой "стратегии Kubernetes", когда руководство организации верит в то, что Kubernetes послужит толчком к модернизации процессов разработки и функционирования ПО. Kubernetes — это замечательная технология, но она не должна определять направление развития вашей организации в отношении инфраструктуры, платформы и/или программного обеспечения. Мы извиняемся, если вам это и так понятно, но вы бы удивились, узнав, сколько исполнительных специалистов и высокопоставленных руководителей, с которыми мы разговаривали, считают, что платформа Kubernetes сама по себе является решением проблем, в то время как на самом деле их проблемы были связаны с доставкой приложений, разработкой программного обеспечения или организационными/кадровыми вопросами. К Kubernetes лучше относиться как к одному из элементов инфраструктуры, который позволяет создавать инфраструктурную платформу для ваших приложений. Мы все ходим вокруг да около идеи о платформе приложений, поэтому давайте поговорим о том, что это такое.

Что такое платформа приложений

Одним из ключевых этапов на нашем пути к эксплуатации должно стать обсуждение идеи о платформе приложений. В нашем понимании *платформа приложений* — это подходящее место для выполнения приложений. Как и с большинством определений в данной книге, его практическое воплощение зависит от конкретной организации. Разные структурные подразделения организации ориентируются на такие желаемые результаты, как удовлетворенность разработчиков, снижение эксплуатационных расходов, сокращение времени цикла обратной связи при доставке ПО и т. д. Платформа приложений зачастую находится на пересечении приложений и инфраструктуры. Удобство разработки (англ. *developer experience* или *devx*) обычно является ключевым принципом в этой области.

Платформы приложений бывают разными. Некоторые в основном инкапсулируют такие основополагающие компоненты, как IaaS (например, AWS) или средство оркестрации (например, Kubernetes). Отличным примером этой модели является Heroku. С помощью данного сервиса вы можете взять проект, написанный на таких языках, как Java, PHP или Go, и развернуть его в реальных условиях эксплуатации с помощью одной команды. Ваше приложение работает в окружении множества сервисов, которые пришлось администрировать самостоятельно, если бы они не предоставлялись самой платформой. Имеются в виду такие вещи, как средства сбора метрик, сервисы для работы с данными и непрерывная доставка (англ. *Continuous*

Delivery или CD). Платформа также дает вам базовые конфигурации для запуска набора наиболее распространенных приложений, которые могут легко масштабироваться. Используется ли Kubernetes в Heroku? Есть ли у Heroku собственные центры обработки данных (ЦОД), или все работает поверх AWS? Какая разница? Для пользователей Heroku эти детали неважны. Важно то, что эти задачи делегируются провайдеру или платформе, что позволяет разработчикам больше времени уделять решению бизнес-проблем. Такой подход применяется не только в облачных сервисах. В OpenShift от RedHat принята похожая модель, в которой Kubernetes является, скорее, одной из деталей реализации, а разработчики и администраторы платформы взаимодействуют с набором более высокоуровневых абстракций.

Почему бы на этом не остановиться? Если такие платформы, как Cloud Foundry, OpenShift и Heroku уже решили эти проблемы за нас, зачем возиться с Kubernetes? Существенным недостатком многих готовых платформ приложений является то, что они навязывают нам свои "взгляды". Делегирование всех обязанностей по управлению и обслуживанию внутренней части системы позволяет избежать большого объема работ по сопровождению. Но в то же время, если методы обнаружения сервисов или управления конфиденциальными данными, которые применяет платформа, не соответствуют вашим организационным требованиям, вы можете быть лишены возможности решить эту проблему. Кроме того, есть такое понятие как зависимость от поставщика или от чужого мнения. Абстракции влекут за собой определенные взгляды на то, как должны проектироваться, упаковываться и развертываться ваши приложения. Это означает, что переход на другую систему может оказаться непростым. Например, рабочие задания намного проще перенести между Google Kubernetes Engine (GKE) и Amazon Elastic Kubernetes Engine (EKS), чем между EKS и Cloud Foundry.

Спектр подходов

Как вы уже сами видите, существует несколько подходов к созданию успешной платформы приложений. Давайте сделаем для демонстрации несколько далеко идущих предположений и сравним теоретические компромиссы, свойственные разным подходам. На рис. 1.3 за пример взята типичная компания среднего/крупного размера, с которой мы обычно имеем дело.

В левой нижней части мы видим развертывание самих кластеров Kubernetes, что требует относительно небольших усилий со стороны инженеров, особенно если за обслуживание управляющего уровня отвечает сторонний сервис вроде EKS. Такой подход обеспечивает не слишком высокую готовность к промышленному внедрению, так как большинство организаций сталкивается с необходимостью в дополнительной работе поверх Kubernetes. Но в некоторых ситуациях одной лишь платформы Kubernetes может быть достаточно, например, при использовании отдельных кластеров.

В правой нижней части находятся более зрелые платформы, которые предоставляют разработчикам полный набор готовых возможностей. Cloud Foundry — отличный пример проекта, который решает множество задач, возлагаемых на платформу

приложений. Программное обеспечение, работающее в Cloud Foundry, не должно выходить за те рамки, которые очертили разработчики этой платформы. А вот платформа OpenShift, которая куда лучше готова к промышленному использованию, чем просто Kubernetes, дает большую свободу выбора относительно того, как ее настраивать. Что собой являет эта гибкость: преимущество или лишнюю работу? Это ключевой вопрос, на который вы должны ответить.

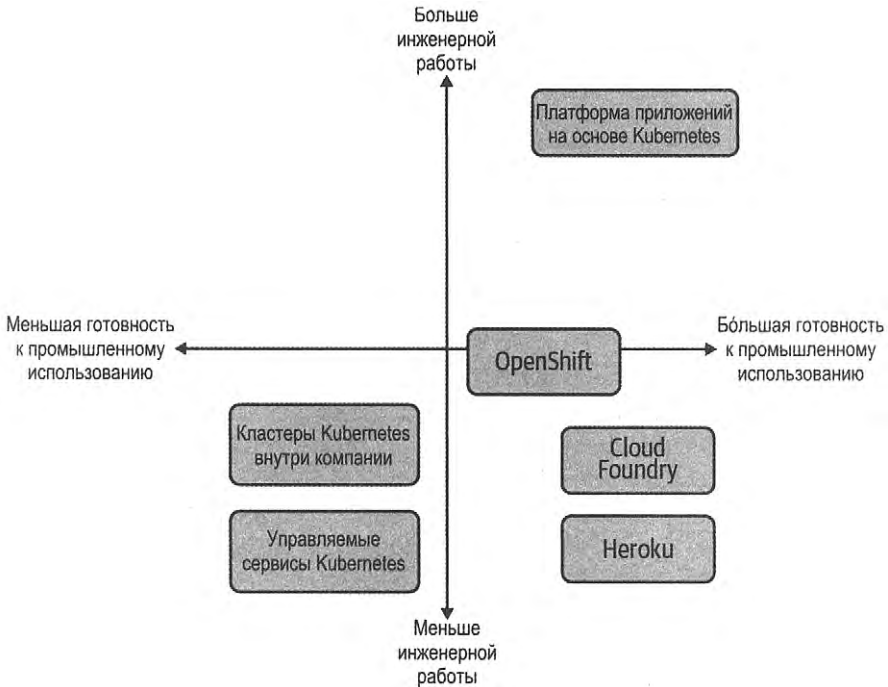


Рис. 1.3. Разнообразные способы организации платформы приложений для разработчиков

Наконец, справа вверху представлено создание платформы приложений поверх Kubernetes. Это, несомненно, требует больше всего инженерных усилий по сравнению с остальными вариантами, по крайней мере, с точки зрения платформы. Но благодаря расширяемости Kubernetes вы можете получить результат, который будет лучше соответствовать вашим потребностям в плане разработки, инфраструктуры и бизнеса.

Спектр подходов с учетом потребностей вашей организации

Диаграмме на рис. 1.3 недостает третьего измерения, оси Z , которая показывает, насколько тот или иной подход соответствует вашим требованиям. Рассмотрим еще одно графическое представление. На рис. 1.4 показано, как это может выглядеть, если учитывать соответствие платформы потребностям организации.

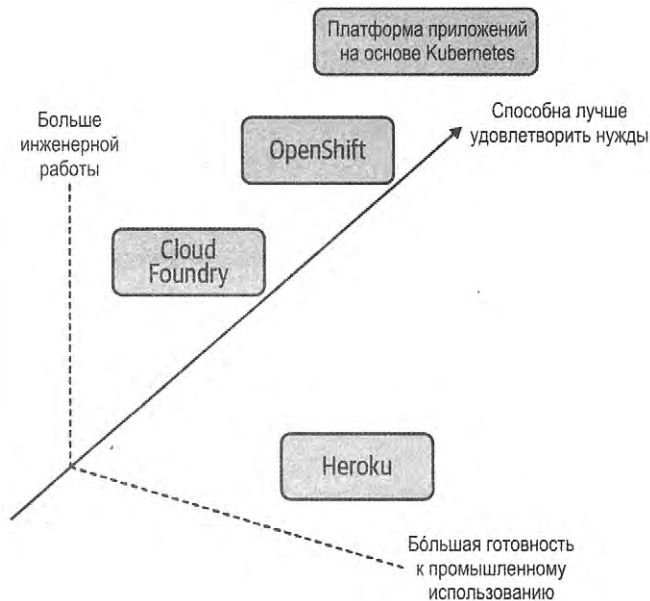


Рис. 1.4. Дополнительное измерение (ось Z), сопоставляющее варианты с потребностями вашей организации

С точки зрения требований, возможностей и ожидаемого поведения, самостоятельное создание платформы почти всегда дает наиболее подходящий результат. Или, по крайней мере, наиболее близкий к таковому. Ведь вы можете создать что угодно! Теоретически вы могли бы воссоздать в своей компании Heroku на основе Kubernetes, с небольшими изменениями функциональности, но при этом следует учитывать соотношение затрат и полученной выгоды (оси X и Y). Давайте добавим в наш разговор больше конкретики и рассмотрим некоторые требования к платформе приложений следующего поколения:

- ◆ регламент обязывает размещать основную часть системы локально;
- ◆ вам нужно поддерживать свои многочисленные физические серверы вместе с ЦОД, в котором используется vSphere;
- ◆ вы хотите удовлетворить растущий спрос со стороны разработчиков на упаковывание приложений в контейнеры;
- ◆ вам нужно создать API-интерфейс для механизмов самообслуживания, которые позволят отойти от выделения инфраструктурных ресурсов "на основе тикетов";
- ◆ вы хотите убедиться в том, что разрабатываемые вами API-интерфейсы не завязаны на конкретного поставщика, так как в прошлом миграция с такого рода систем стоила вам миллионы;
- ◆ вы не против платить за корпоративную поддержку разнообразных продуктов в своем стеке, но не в восторге от моделей, в которых весь стек лицензируется для каждого узла, ядра или экземпляра приложения.

Чтобы определить, является ли построение платформы приложений разумным начинанием, мы должны понимать, насколько "созрели" наши инженерные возможности, есть ли у нас желание заниматься формированием и развитием групп разработчиков, и обладаем ли мы достаточными ресурсами.

Платформы приложений: подводим итоги

Следует признать, что определение платформы приложений остается довольно размытым. Мы сосредоточились на разнообразных платформах, которые, как нам кажется, дают командам разработчиков нечто большее, чем просто оркестрацию приложений. Мы также ясно дали понять, что Kubernetes можно изменять и дополнять для получения похожих результатов. Выводя свое мышление за рамки того, как внедрить Kubernetes, и задаваясь вопросами наподобие "как сейчас организован рабочий процесс разработчиков, какие у них проблемы и желания?", группы сотрудников, которые занимаются платформой и инфраструктурой, будут более успешными в своей профессиональной деятельности. Можно утверждать, что, сосредоточившись на последнем, вы с большей вероятностью наметите себе подходящий путь к эксплуатации и справитесь с непростым процессом внедрения. В конечном счете мы хотим удовлетворить требования к инфраструктуре, безопасности и удобству разработки, чтобы наши клиенты (которыми обычно являются разработчики) получили решение, отвечающее их потребностям. Как правило, мы не стремимся предоставить "мощный" движок, поверх которого каждый разработчик должен создать собственную платформу, что с юмором продемонстрировано на рис. 1.5.



Рис. 1.5. Если разработчики хотят иметь готовое решение (например, полноценный автомобиль), не надейтесь, что им будет достаточно двигателя без кузова, колес и прочего

Создание платформ приложений на основе Kubernetes

Итак, мы воспринимаем Kubernetes как один из этапов нашего пути к развертыванию. Учитывая это, справедливым будет следующий вопрос: "Может, у Kubernetes просто недостаточно возможностей?" Проект Kubernetes был вдохновлен одним из важнейших принципов философии Unix: "пишите программы, которые делают что-то одно и делают это хорошо". Мы считаем, что лучшими возможностями этой платформы являются те, которых у нее нет! Это становится еще очевидней после

мучений с универсальными платформами, которые пытаются решать за вас все проблемы человечества. Блестящая черта платформы Kubernetes — ее узкая направленность: она пытается быть хорошим средством оркестрации, но в то же время предоставляет четкие интерфейсы, поверх которых можно написать что-то свое. Таким образом, платформу можно сравнить с фундаментом дома.

Хороший фундамент должен быть прочным, подходить для возведения остальных конструкций и предоставлять подходящие "интерфейсы" для проведения коммуникаций. Несмотря на важную роль, сам по себе фундамент редко когда становится подходящим местом для проживания. Обычно на фундаменте должен находиться какой-то дом. Прежде чем обсуждать *строительство* поверх такого фундамента, как Kubernetes, давайте рассмотрим готовое меблированное жилище, показанное на рис. 1.6.

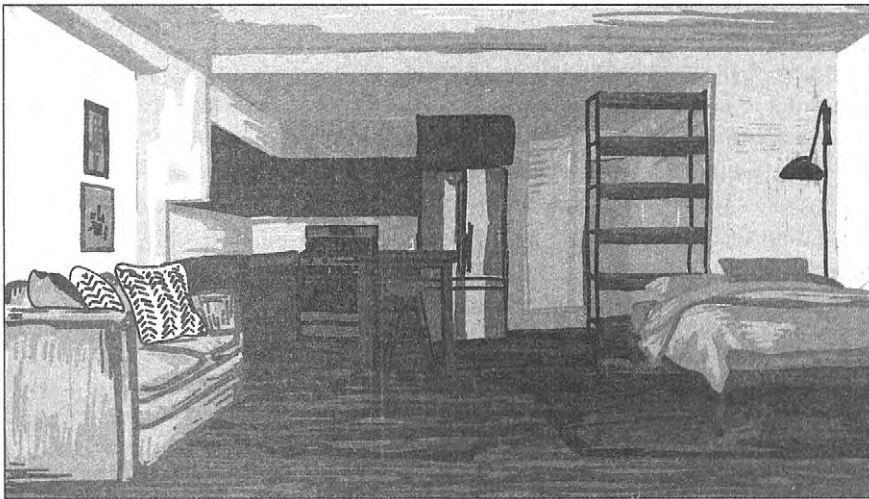


Рис. 1.6 Апартаменты, готовые к заселению. Похожи на PaaS, такие как Heroku (иллюстрация Джессики Аппельбаум)

Этот вариант жилья, подобный нашим примерам с Heroku, подходит для проживания и не требует дополнительных вложений. Вы, конечно, можете немного изменить интерьер, но большинство проблем уже решено. Если подходит арендная плата, и вы согласны смириться с тем, как все устроено внутри, вы можете жить припеваючи с самого первого дня.

Но вернемся к платформе Kubernetes, которую мы сравнили с фундаментом. Поверх нее можно построить жилой дом, как показано на рис. 1.7.

Инвестируя в планирование, проектирование и поддержку, мы можем создавать замечательные платформы, способные выполнять рабочие задания в разных организациях. Это означает, что мы контролируем каждый аспект конечного результата. Дом может и должен быть приспособлен к нуждам его будущих жильцов (наших приложений). Давайте теперь разберем различные слои и соображения, благодаря которым это возможно.

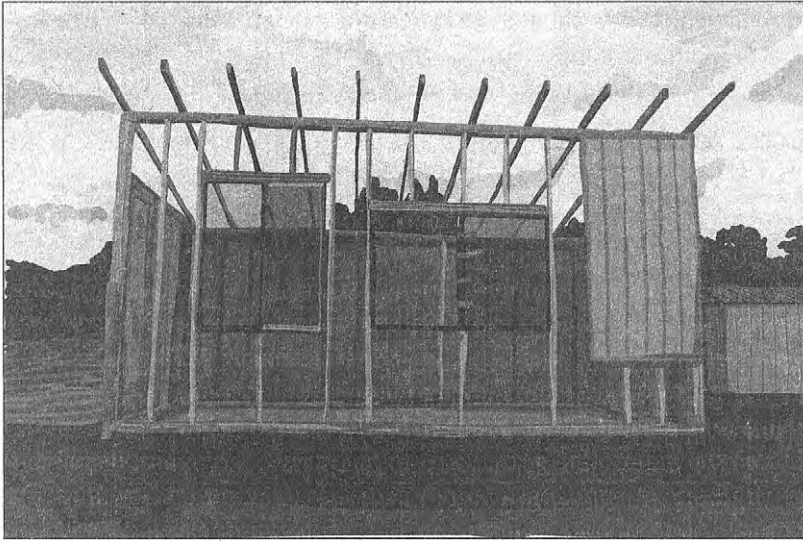


Рис. 1.7. Строительство дома. Похоже на создание платформы приложений, в основе которой лежит Kubernetes (иллюстрация Джессики Аппельбаум)

Начиная снизу

Мы должны начать с самого низа, в том числе с того, какие технологии будет использовать Kubernetes. Обычно речь идет о центре обработки данных или облачном провайдере, который предлагает вычислительные ресурсы, хранилище и доступ к сети. Дальше на основе этого можно развертывать Kubernetes. Всего за несколько минут вы можете получить кластер, который работает поверх внутренней инфраструктуры. Kubernetes можно развернуть несколькими способами, которые мы подробно обсудим в *главе 2*.

Следующий важный аспект существования кластеров Kubernetes, который поможет нам определить, что следует создавать поверх них, — процесс внедрения Kubernetes. Его ключевые элементы представлены на рис. 1.8.

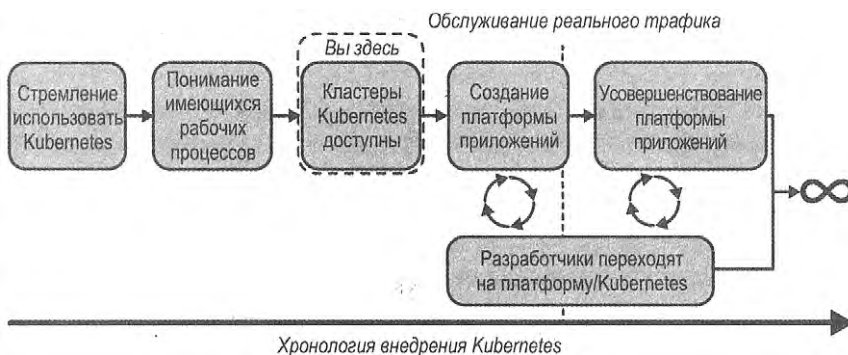


Рис. 1.8. Процесс, через который могут пройти наши команды разработчиков на пути к полноценному внедрению Kubernetes

С появлением Kubernetes в вашей организации, не удивляйтесь, если к вам начнут обращаться с подобными вопросами:

- ◆ "Как сделать так, чтобы трафик между приложениями шифровался?"
- ◆ "Как сделать так, чтобы исходящий трафик проходил через шлюз, который гарантирует CIDR?"
- ◆ "Как организовать трассировку и дашборды для приложений?"
- ◆ "Как проводить погружение разработчиков, не требуя от них глубоких знаний Kubernetes?"

Этот список можно продолжать бесконечно. Зачастую нам самим приходится определять, какие требования должны удовлетворяться на уровне платформы, а какие — на уровне приложения. Главное здесь — иметь глубокое понимание существующих рабочих процессов, чтобы создаваемая нами платформа отвечала текущим ожиданиям. Если мы не в состоянии обеспечить нужную функциональность, как это повлияет на группы разработчиков?

Дальше мы можем приниматься за создание платформы поверх Kubernetes. Очень важно, чтобы в ходе данного процесса мы действовали совместно с разработчиками, желающими опробовать нашу платформу на ранних этапах, и знали об их впечатлениях, поскольку это поможет нам принимать обоснованные решения с учетом оперативно поступающих отзывов. Такой процесс не должен прекращаться даже после развертывания платформы в промышленном окружении. Не рассчитывайте на то, что платформа получится статической, и что разработчики будут пользоваться ею на протяжении десятилетий. Чтобы достичь успеха, мы должны постоянно взаимодействовать с нашими группами разработчиков, что позволит нам узнавать о проблемах и недостающих возможностях, которые могли бы ускорить темпы разработки. Для начала будет неплохо подумать о том, какой уровень взаимодействия с Kubernetes мы должны ожидать от наших разработчиков. В итоге мы уясним, насколько общими должны быть наши абстракции.

Спектр абстрагирования

В прошлом нам встречались несколько надменных высказывания вроде "Если вашим разработчикам известно о том, что они используют Kubernetes, это провал!" Такой взгляд на взаимодействие с Kubernetes может быть вполне приемлемым, особенно если вы создаете продукты или услуги, потребителям которых все равно, какая технология оркестрации находится внутри. Например, вы можете работать над системой управления базами данных, которая поддерживает разные БД. Ваших разработчиков, скорее всего, не интересует, внутри чего выполняются сегменты или экземпляры базы данных: Kubernetes, Bosh или Mesos! Однако делать эту философию основным критерием успеха вашей команды будет опасно. По мере создания поверх Kubernetes платформы с сервисами для лучшего обслуживания наших клиентов нам неоднократно придется принимать решения о том, как должны выглядеть наши абстракции. Это проиллюстрировано на рис. 1.9.

Данный вопрос может не давать покоя тем, кто занимается созданием платформы. Абстракции приносят много пользы. Такие проекты, как Cloud Foundry, предостав-

ляют разработчикам полнофункциональное окружение, например, с помощью одной единственной команды `cf push` мы можем взять приложение, собрать его и вернуть для обслуживания реального трафика. Сосредоточившись на достижении этой цели и описанных возможностях, Cloud Foundry углубляет свою поддержку работы поверх Kubernetes, и мы ожидаем, что этот переход будет выглядеть, скорее, как аспект реализации, чем как изменение функциональности. Еще одна закономерность, которую мы наблюдаем, состоит в желании предлагать компаниям нечто большее, чем просто Kubernetes, но при этом не вынуждать разработчиков делать сознательный выбор между разными технологиями. Например, в некоторых компаниях параллельно с Kubernetes используется Mesos. Такие компании создают абстракцию, позволяющую прозрачно выбирать место размещения приложений, не перекладывая эту обязанность на разработчиков, что также позволяет им избежать зависимости от конкретной технологии. Противоположный подход — создание абстракции поверх двух систем, которые работают по-разному. Это требует от компании зрелости в принятии решений и существенных инженерных усилий. К тому же, несмотря на то, что разработчикам больше не нужно знать, как взаимодействовать с Kubernetes или Mesos, они должны уметь работать с абстрагированной системой, существующей только в их компании. В наш век открытого исходного кода разработчики, какой бы стек технологий они ни использовали, не очень-то любят изучать системы, опыт работы с которыми будет бесполезен в других организациях.



Рис. 1.9. Противоположные концы спектра: предоставление каждой команде отдельного кластера Kubernetes или полная инкапсуляция Kubernetes в виде платформы как сервиса (англ. Platform as a Service или PaaS)

Наконец, еще одна ловушка, которая нам встречалась, — "одержимость абстракциями", которая не дает предоставить доступ к ключевым возможностям Kubernetes. Со временем это может превратиться в игру в кошки-мышки, когда мы пытаемся не отставать от проекта, в результате чего наша абстракция может стать такой же сложной, как и система, которую мы абстрагируем.

На противоположном конце спектра находятся создатели платформ, которые хотят предложить разработчикам кластеры с самообслуживанием, которые тоже могут быть отличной моделью. Ответственность за взаимодействие с Kubernetes перекладывается на команды разработчиков. Понимают ли они, как работают Развертывания (Deployments), Наборы реплик (Replicas), Pod, Сервисы и API-интерфейсы объектов Ingress? Умеют ли они обращаться с единицами измерения вычислительных ресурсов, milliCPU, и знают ли они, как происходит выделение недоступных ресурсов (overcommit)? Известно ли им, как добиться того, чтобы выполнение приложений, сконфигурированных с более чем одной репликой, всегда планировалось на разных узлах? Если да, то это идеальная возможность избежать чрезмерного ус-