Оглавление

Содержание

Предисловие

Мы написали эту книгу для всех, кто хочет получить новые знания о K8s (Kubernetes) и сразу же углубиться в различные темы, связанные с хранением, сетевыми взаимодействиями и использованием разнообразных инструментов.

Мы не ставили перед собой цель написать исчерпывающее руководство по всем возможностям K8s API (это просто невозможно), но искренне верим, что, прочитав эту книгу, пользователи обретут понимание, которое поможет им по-новому взглянуть на сложные задачи организации инфраструктуры в промышленных кластерах и увидеть общее развитие ландшафта Kubernetes в более широком контексте.

Конечно, есть немало книг, позволяющих пользователям изучить основы Kubernetes, но мы хотели написать книгу, рассказывающую об основных технологиях, составляющих Kubernetes. Здесь мы расскажем вам все тонкости организации сети и плоскости управления, а также некоторые другие темы, чтобы вы смогли понять внутренние особенности работы Kubernetes. Понимание этих деталей сделает вас лучшим инженером DevOps и программистом.

Мы также надеемся вдохновить новых пользователей Kubernetes. Свяжитесь с нами в Твиттере (@jayunit100, @chrislovcnm), если решите принять участие в жизни сообщества Kubernetes или помочь нам добавить больше примеров для этой книги.

О книге

Кому адресована эта книга

Всем желающим узнать больше о внутреннем устройстве Kubernetes, о том, как рассуждать о его режимах отказа и возможности расширения под нужды пользователей. Если вы не знаете, что такое Pod, то, конечно, можете приобрести эту книгу, но прежде прочтите какую-нибудь другую книгу, где вы сможете поближе познакомиться с терминологией.

Также книга пригодится операторам, желающим общаться на едином языке с сотрудниками ИТ-отделов, техническими директорами и другими руководителями о том, как внедрить Kubernetes, сохранив при этом основные принципы построения инфраструктуры, существовавшие до появления контейнеров. Эта книга действительно поможет преодолеть разрыв между новыми и старыми решениями по проектированию инфраструктуры. По крайней мере, мы на это надеемся!

Организация книги

Эта книга состоит из 15 глав:

- глава 1. Дает общий обзор Kubernetes для новичков;
- глава 2. Рассматривает идею модуля Pod как атомарного строительного блока для приложений и закладывает основы для последующих глав, подробно рассматривающих низкоуровневые детали Linux;
- глава 3. Углубляется в детали использования в Kubernetes низкоуровневых примитивов Linux для реализации концепций более высокого уровня, включая модули Pod;
- глава 4. Здесь начинается изучение внутренних особенностей процессов и их изоляции в Linux, которые являются одними из менее известных деталей ландшафта Kubernetes;
- глава 5. После подробного знакомства с модулями Pod углубляется в организацию сетевых взаимодействий между ними и рассказывает, как они соединяются друг с другом через разные узлы;

- глава 6. Вторая глава, посвященная сетевым взаимодействиям, описывающая более широкие аспекты работы модулей Pod и прокси-сервера (`kube-proxy`), а также приемы их настройки и сопровождения;
- глава 7. Первая глава, посвященная вопросам организации хранилища. Здесь дается широкое введение в теоретические основы хранилищ Kubernetes, контейнерный интерфейс хранилища (Container Storage Interface, CSI) и его взаимодействия с kubelet;
- глава 8. Вторая глава, посвященная вопросам организации хранилища. Здесь рассматриваются некоторые более практические аспекты хранения данных, включая особенности emptyDir, Secrets и PersistentVolumes/Dynamic storage;
- глава 9. Углубляется в kubelet и рассматривает некоторые детали запуска модулей Pod и управления ими, включая такие понятия, как CRI, жизненный цикл узла и ImagePullSecrets;
- глава 10. DNS в Kubernetes – сложный механизм, используемый почти всеми контейнерными приложениями для локального доступа к внутренним сервисам. Здесь рассматривается CoreDNS – реализация сервиса DNS для Kubernetes – и порядок выполнения DNS-запросов разными модулями Pod;
- глава 11. Подробно обсуждает плоскость управления, упоминавшуюся в предыдущих главах, включая тонкости работы планировщика, диспетчера контроллеров и сервера API. Они образуют «мозг» Kubernetes и объединяют вместе все низкоуровневые концепции, обсуждавшиеся в предыдущих главах;
- глава 12. После обсуждения логики плоскости управления мы углубимся в etcd, надежный механизм консенсуса для Kubernetes, и особенности его настройки для удовлетворения потребностей плоскости управления Kubernetes;
- глава 13. Представляет обзор NetworkPolicies, RBAC и безопасности на уровне модулей Pod и узлов, о которых должен знать каждый администратор. В этой главе также обсуждается общее развитие политик безопасности модулей Pod;
- глава 14. Здесь рассматривается настройка безопасности на уровне узла и облака, а также другие аспекты безопасности Kubernetes, ориентированные на инфраструктуру;
- глава 15. Эта заключительная глава дает общий обзор прикладных инструментов на примере Carvel, набора инструментов для управления файлами YAML, создания приложений и долгосрочного управления жизненным циклом приложений.

О примерах программного кода

Для этой книги мы подготовили несколько примеров, которые вы найдете в репозитории GitHub (https://github.com/jayunit100/k8sprototypes/), в том числе примеры:

- использования `kind` для настройки реалистичной сети в локальных кластерах с помощью Calico, Antrea или Cillium;
- анализа метрик Prometheus в реальном мире;
- создания приложений с помощью набора инструментов Carvel;
- различных экспериментов, связанных с RBAC.

Эта книга также содержит множество примеров программного кода. Они оформлены `шрифтом фиксированной ширины`, чтобы вам было проще отличать его от основного текста.

Во многих случаях исходный код переформатирован, чтобы уместить его по ширине книжной страницы. В частности, мы добавили разрывы строк и отступы. В редких случаях даже этого было недостаточно, и мы добавили маркеры продолжения строки (➥). Многие листинги сопровождаются комментариями в тексте, подчеркивающими важные понятия. Получить выполняемые фрагменты кода можно из электронной версии книги по адресу https://livebook.manning.com/book/core-kubernetes и в репозитории GitHub https://github.com/jayunit100/k8sprototypes/.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно от-

носятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторах

Джей Вьяс, д-р наук (Jay Vyas, PhD), в настоящее время – штатный инженер в VMware. Работал над несколькими коммерческими дистрибутивами и платформами Kubernetes с открытым исходным кодом, включая OpenShift, VMware Tanzu, внутренними коллективными платформами Black Duck для Kubernetes и установкой Kubernetes для клиентов его консалтинговой компании Rocket Rudolf, LLC. В течение нескольких лет был членом комитета по управлению проектами (Project Management Committee, PMC) в Apache Software Foundation, где работал над несколькими проектами в области больших данных. Он был связан с Kubernetes на различных должностях с момента его создания и в настоящее время уделяет большое внимание сообществам SIG-Windows и SIG-Network. Начинал с создания распределенных систем, одновременно защитив докторскую диссертацию по витринам данных в сфере биоинформатики (объединявшим базы данных в платформы для анализа человеческих и вирусных белковых карт – протеомов). Это привело его в мир больших данных и масштабируемых систем обработки данных и, наконец, в Kubernetes.

Связаться с Джеем можно по адресу @jayunit100 в Твиттере, если вы заинтересованы в сотрудничестве… по какой угодно теме. Уделяет большое внимание спорту, ежедневно пробегает одну милю в спринтерском темпе и подтягивается до отказа. Также увлекается музыкой и имеет несколько синтезаторов, в том числе Prophet-6, который звучит как космический корабль.

Крис Лав (Chris Love) – сертифицированный сотрудник Google Cloud и соучредитель Lionkube. Больше 25 лет занимался разработкой программного обеспечения в таких компаниях, как Google, Oracle, VMware, Cisco, Johnson & Johnson и др. Как идейный лидер в Kubernetes и в сообществе DevOps, Крис Лав участвовал во многих проектах с открытым исход-

ным кодом, включая Kubernetes, kops (в должности руководителя AWS SIG), Bazel (внес вклад в разработку правил для Kubernetes) и Terraform (один из первых разработчиков плагина VMware). В число его профессиональных интересов входят: трансформация ИТ-культуры, технологии контейнеризации, методы и средства автоматизированного тестирования, Kubernetes, Golang (он же Go) и другие языки программирования. Лав обожает заниматься популяризацией DevOps, Kubernetes и технологий, а также обучать людей в сфере ИТ и программного обеспечения.

Вне работы любит кататься на лыжах, играть в волейбол, заниматься йогой и участвовать в мероприятиях на свежем воздухе. Кроме того, вот уже 20 лет занимается боевыми искусствами.

Если вы решите пообщаться с Крисом и задать ему свои вопросы, то сможете связаться с ним по адресу @chrislovenm в Twitter или LinkedIn.

1 Почему появился Kubernetes

В этой главе:

- почему появился Kubernetes;
- основные термины Kubernetes;
- конкретные примеры использования Kubernetes;
- высокоуровневые функции Kubernetes;
- когда нежелательно использовать Kubernetes.

Kubernetes – это платформа с открытым исходным кодом для размещения контейнеров и определения прикладных API для управления облачной семантикой обеспечения этих контейнеров хранилищами данных, сетевыми услугами, поддержкой безопасности и другими ресурсами. Kubernetes обеспечивает непрерывную синхронизацию всего пространства состояний ваших приложений, в том числе способов доступа к ним из внешнего мира.

Зачем внедрять Kubernetes в свое окружение? Не проще ли выделить все необходимые ресурсы вручную с помощью инструмента управления инфраструктурой, связанного с DevOps? Ответ зависит от того, как мы определяем процесс DevOps и его интеграцию в общий жизненный цикл приложения. DevOps продолжает расширяться и включает в себя процессы, инженеров и инструменты, которые поддерживают более автоматизированное администрирование приложений в центре обработки данных. Одно из условий успешного

решения этой задачи – воспроизводимость инфраструктуры: изменение, внесенное для устранения инцидента в одном компоненте, которое не воспроизводится полностью во всех других идентичных компонентах, означает, что один или несколько компонентов отличаются.

В этой книге мы подробно рассмотрим передовые практики использования Kubernetes в DevOps, обеспечивающие репликацию компонентов по мере необходимости и уменьшение частоты сбоев системы. Мы также исследуем внутренние процессы, чтобы лучше понять, как работает Kubernetes и как с его применением получить максимально эффективную систему.

1.1 Предварительный обзор некоторых основных терминов

В 2021 году Kubernetes стала одной из наиболее широко используемых облачных технологий. Из-за этого мы не всегда полностью определяем новые термины, прежде чем ссылаться на них. Для новичков в Kubernetes или недостаточно хорошо знакомых с некоторыми терминами далее мы приводим несколько ключевых определений, к которым вы можете периодически обращаться, читая несколько первых глав этой книги и осваивая эту новую вселенную. Мы определим эти понятия более подробно и в более широком контексте, когда углубимся в них позже в этой книге:

- *CNI* (*Container Networking Interface*) и *CSI* (*Container Storage Interface*) – сетевой интерфейс контейнеров и интерфейс хранилища для контейнеров соответственно; позволяют подключать к сетям и хранилищам модули Pod (с контейнерами), работающие в Kubernetes;
- *контейнер* (*Container*) – образ Docker или OCI (Open Container Initiative), который обычно запускает приложение;
- *плоскость управления* (*Control plane*) – мозг кластера Kubernetes, осуществляющий планирование контейнеров и управляющий всеми объектами Kubernetes (которые иногда называют мастер-объектами);
- *набор демонов* (*DaemonSet*) – аналог развертывания (Deployment), но выполняется на каждом узле кластера;
- *развертывание* (*Deployment*) – набор модулей, которыми управляет Kubernetes;
- *kubectl* – инструмент командной строки для взаимодействия с панелью управления Kubernetes;
- `kubelet` – агент Kubernetes, работающий на узлах кластера. Обеспечивает поддержку плоскости управления;
- *узел* (*Node*) – машина, на которой запущен процесс kubelet;

- *OCI* (*Open Container Initiative*) – общий формат образа для создания выполняемых автономных приложений. Также называется *образами Docker*;
- *Pod* (*модуль*) – объект Kubernetes, инкапсулирующий действующий контейнер.

1.2 Проблема дрейфа инфраструктуры и Kubernetes

Управление инфраструктурой – это воспроизводимый способ управления «дрейфом» конфигурации этой инфраструктуры по мере изменения аппаратного обеспечения, нормативов и других требований, действующих в центре обработки данных. Это относится и к *определению* приложений, и к *управлению* хостами, на которых выполняются эти приложения. ИТ-инженеры слишком хорошо знакомы с типичными проблемами, такими как:

- обновление версии Java на нескольких серверах;
- выявление причин отказа некоторых приложений в определенных местах;
- замена или масштабирование старого или сломанного оборудования и перенос приложений;
- ручное управление маршрутами для балансировки нагрузки;
- отсутствие описания новых изменений инфраструктуры в документации из-за отсутствия общего обязательного языка конфигурации.

В процессе управления и обновления серверов в центре обработки данных или в облаке увеличивается вероятность «отклонения» их исходных определений от предполагаемой архитектуры. Приложения могут запускаться в неправильных местах, с неправильным набором ресурсов или с доступом к неправильным хранилищам.

Kubernetes дает возможность централизовать управление пространством состояний всех приложений с использованием одного удобного инструмента: `kubectl` (https://kubernetes.io/docs/tasks/tools/) – клиента командной строки, выполняющего вызовы REST API к серверу Kubernetes API. Также есть возможность использовать клиентов Kubernetes API для выполнения этих же задач программно. Установить `kubectl` и протестировать его в кластере довольно просто, что мы и сделаем в начале этой книги.

Предыдущие подходы к управлению сложным пространством состояний приложений основывались на таких технологиях, как Puppet, Chef, Mesos, Ansible и SaltStack. Kubernetes заимствовал лучшие черты этих подходов и использует возможности управления состоянием таких инструментов, как Puppet, а также идеи некоторых приложений

и примитивов планирования, предоставляемых таким программным обеспечением, как Mesos.

Ansible, SaltStack и Terraform играли важную роль в настройке инфраструктуры (определяя требования, специфичные для ОС, такие как брандмауэры или установка двоичных файлов). Kubernetes тоже поддерживает эту идею, но использует *привилегированные контейнеры* в среде Linux (в Windows v1.22 они известны как *HostProcess Pods*). Например, привилегированный контейнер в системе Linux может управлять правилами iptables для организации маршрутизации трафика к приложениям, что, собственно, и делает прокси-сервер Kubernetes Service (известный как *kube-proxy*).

Google, Microsoft, Amazon, VMware и многие другие компании взяли на вооружение контейнеризацию как основную стратегию, позволяющую клиентам запускать сотни и тысячи приложений в различных облачных окружениях и окружениях без системного программного обеспечения. Соответственно, контейнеры оказываются фундаментальным примитивом *и* для запуска приложений, *и* для управления инфраструктурой (например, для предоставления контейнерам IP-адресов), которые запускают сервисы, необходимые приложениям (например, специализированные хранилища или брандмауэры с определенными настройками), и, что особенно важно, сами приложения.

Kubernetes (на момент написания этой книги) практически бесспорно считается современным стандартом для организации и запуска контейнеров в любом облачном окружении, на сервере или в центре обработки данных.

1.3 Контейнеры и образы

Приложения имеют зависимости, которые должны удовлетворяться хостом, на котором они выполняются. В доконтейнерную эпоху разработчики решали эту задачу в произвольном порядке (например, приложению Java требовалось действующая виртуальная машина JVM вместе с настроенным брандмауэром, поддерживающим возможность доступа к базе данных).

По своей сути Docker можно рассматривать как способ запуска контейнеров, где *контейнер* – это работающий образ OCI (https://github.com/opencontainers/image-spec). Спецификация OCI – это стандартный способ определения образа, который может быть запущен такой программой, как Docker, и в конечном счете представляет собой архив с различными слоями. Слои в архиве образа содержат такие компоненты, как двоичные файлы Linux и файлы приложений. Соответственно, когда вы запускаете контейнер, среда выполнения контейнеров (такая как Docker, containerd или CRI-O) берет образ, распаковывает его и запускает процесс в хост-системе, который, в свою очередь, запускает содержимое образа.

Контейнеры добавляют слой изоляции, устраняющий необходимость управления библиотеками на сервере или предварительной загрузки инфраструктуры другими зависимостями приложений (рис. 1.1). Например, если есть два приложения Ruby, которым требуются разные версии одной и той же библиотеки, то можно использовать два контейнера. Каждое приложение Ruby будет изолировано внутри своего контейнера и использовать определенную версию библиотеки.

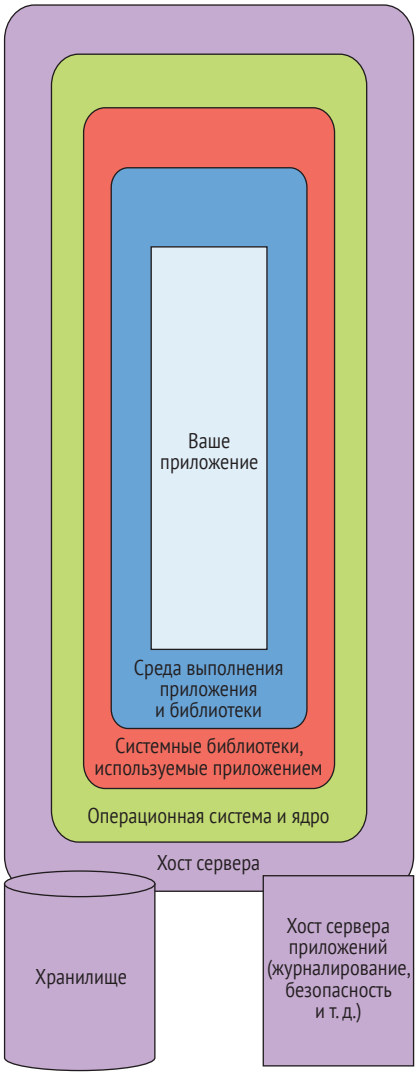

Рис. 1.1 Приложения, выполняющиеся в контейнерах

Разработчикам хорошо известна фраза: «Ну, это работает на моей машине». Программное обеспечение часто может работать в одном окружении или на одном компьютере, но не работать в другом. Обра-

зы упрощают запуск одного и того же программного обеспечения на разных серверах. Подробнее об образах и контейнерах мы поговорим в главе 3.

Объединение образов с платформой Kubernetes, позволяющей запускать неизменяемые серверы, дает небывалый уровень простоты. Поскольку контейнеры быстро становятся отраслевым стандартом развертывания программных приложений, стоит привести некоторые цифры:

- *по результатам опроса более 88 000 разработчиков, Docker и Kubernetes заняли третье место среди самых популярных технологий разработки в 2020 году*, сразу после Linux и Docker (http://mng.bz/nY12);
- *служба Datadog недавно обнаружила, что Docker охватывает 50 или более процентов рабочего процесса среднего разработчика.* Аналогично более 25 % компаний полностью перешли на использование Docker (https://www.datadoghq.com/docker-adoption/).

Однако использование контейнеров немыслимо без автоматизации, и именно этой цели служит Kubernetes. Kubernetes доминирует в пространстве контейнеров так же, как доминировали в своих сферах СУБД Oracle и платформа виртуализации vSphere во времена расцвета. Спустя годы базы данных Oracle и vSphere все еще пользуются большой популярностью; такое же долголетие мы прогнозируем и для Kubernetes.

Мы начнем эту книгу знакомством с базовыми особенностями Kubernetes. Ее цель в том, чтобы вывести вас за пределы базовых принципов и познакомить с низкоуровневым ядром. Давайте начнем наше погружение и посмотрим на чрезвычайно упрощенный рабочий процесс Kubernetes (также называемый «K8s»), демонстрирующий, как некоторые из пользователей создают и запускают микросервисы.

1.4 Базовая основа Kubernetes

Все сущее в Kubernetes определяется в виде простых текстовых файлов в формате YAML или JSON, и платформа запускает образы OCI декларативным способом. Тот же подход (с текстовыми файлами в формате YAML или JSON) можно использовать для настройки сетевых правил, аутентификации и авторизации на основе ролей (RBAC) и т. д. Изучив один синтаксис и его структуру, можно настраивать, управлять и оптимизировать любые системы Kubernetes.

Давайте рассмотрим короткий пример запуска простого приложения в Kubernetes. Не волнуйтесь, если что-то покажется вам непонятным; далее в книге мы рассмотрим еще множество реальных примеров, которые проведут нас через весь жизненный цикл приложения. Считайте, что это всего лишь наглядная демонстрация наших пассов руками, которые мы делали до сих пор. Начнем с конкретного примера – микросервиса. Следующий фрагмент кода генерирует файл Dockerfile, который определяет образ для запуска MySQL:

```
FROM alpine:3.15.4
RUN apk add --no-cache mysql
ENTRYPOINT ["/usr/bin/mysqld"]
```

Обычно этот образ создается (с помощью `docker build`) и сохраняется (с помощью, например, `docker push`) в *реестре OCI* (место, где образ может храниться и извлекаться контейнером в момент запуска). Общедоступный реестр с открытым исходным кодом, в котором вы можете размещать свои образы, доступен по адресу https://github.com/goharbor/harbor. Еще один похожий реестр, тоже широко используемый для хранения миллионов образов приложений, находится по адресу https://hub.docker.com/. Для целей нашего примера предположим, что мы отправили образ в реестр и теперь запускаем его. Нам также может понадобиться создать контейнер для взаимодействия с этой службой (например, с приложением на Python, которое пользуется базой данных MySQL). Мы могли бы определить его образ Docker так:

```
FROM python:3.7
WORKDIR /myapp
COPY src/requirements.txt ./
RUN pip install -r requirements.txt
COPY src /myapp
CMD [ "python", "mysql-custom-client.py" ]
```

Чтобы запустить клиента и сервер MySQL в виде контейнеров в окружении Kubernetes, нужно создать два объекта типа Pod. Каждый из них может выполнять один из контейнеров, например:

```
apiVersion: v1
kind: Pod
metadata:
  name: core-k8s
  spec:
  containers:
    - name: my-mysql-server
      image: myregistry.com/mysql-server:v1.0
---
apiVersion: v1
kind: Pod
metadata:
  name: core-k8s-mysql
  spec:
  containers:
    - name: my-sqlclient
      image: myregistry.com/mysql-custom-client:v1.0
      command: ['tail','-f','/dev/null']
```

Обычно такие фрагменты YAML сохраняются в текстовых файлах (например, myapp.yaml) и выполняются с помощью клиента Kubernetes (например, `kubectl create -f my-app.yaml`). Этот инструмент под-

ключается к серверу Kubernetes API и передает определение в формате YAML для сохранения. Затем Kubernetes автоматически извлекает определения двух модулей Pod, имеющиеся на сервере API, и проверяет, где они должны быть запущены.

Это происходит не мгновенно, потому что узлам в кластере требуется время, чтобы среагировать на постоянно происходящие события и обновить состояние своих объектов Node через агента kubelet, взаимодействующего с сервером API. Также требуется, чтобы образы OCI присутствовали и были доступны для узлов в кластере Kubernetes. В любой момент может что-то пойти не так, поэтому мы называем Kubernetes системой, *стабильной в конечном счете*, в которой согласование желаемого состояния с течением времени является ключевой философией дизайна. Эта модель согласованности (по сравнению с моделью гарантированной согласованности) обеспечивает возможность постоянного мониторинга изменений в общем пространстве состояний всех приложений в кластере и позволяет платформе Kubernetes определять, *как* эти приложения приводятся в движение с течением времени.

Этот подход естественным образом укладывается в сценарии реального мира. Например, если вам нужно, чтобы «пять приложений были распределены по трем зонам в облаке», этого легко добиться, определив несколько строк YAML с примитивами планирования Kubernetes. Конечно, вам придется гарантировать существование этих трех зон и их доступность для планировщика, но, даже если вы этого не сделаете, Kubernetes, по крайней мере, запланирует некоторые рабочие нагрузки в доступных зонах.

Проще говоря, Kubernetes позволяет определить желаемое состояние всех приложений в кластере, их подключение к сети, место работы, используемое хранилище и т. д., делегируя базовую реализацию этих деталей самой платформе Kubernetes. Соответственно, вам редко придется выполнять однократное обновление Ansible или Puppet в сценарии промышленного использования Kubernetes (если только вы не переустанавливаете саму платформу Kubernetes, но даже в этом случае можно воспользоваться такими инструментами, как Cluster API, позволяющими управлять платформой Kubernetes с помощью самой Kubernetes (как бы не запутаться во всем этом)).

1.4.1 Все инфраструктурные правила в Kubernetes определяются в обычных файлах YAML

Kubernetes автоматизирует все аспекты стека с помощью Kubernetes API, которым можно полностью управлять как ресурсами YAML и JSON. К их числу относятся традиционные правила инфраструктуры (которые так или иначе применяются к микросервисам), такие как:

- конфигурация портов или IP-маршрутов;
- постоянная доступность хранилища для приложений;

- размещение программного обеспечения на определенных или произвольных серверах;
- обеспечение безопасного доступа приложений друг к другу с использованием, например, RBAC или сетевых правил;
- конфигурация DNS для каждого приложения и глобально.

Все эти компоненты определяются в конфигурационных файлах, представляющих объекты в Kubernetes API. Kubernetes использует эти стандартные блоки и контейнеры, применяет изменения, отслеживает эти изменения и устраняет сбои или нарушения, пока не будет достигнуто желаемое конечное состояние. Когда «ночью что-то идет не так», Kubernetes автоматически обрабатывает множество сценариев и избавляет нас от решения проблем вручную. Правильная настройка сложных систем с применением средств автоматизации позволяет команде DevOps сосредоточиться на решении важных задач, планировать будущее и находить лучшие в своем классе решения для бизнеса. Давайте далее рассмотрим некоторые возможности, предлагаемые платформой Kubernetes для поддержки модулей Pod.

1.5 Возможности Kubernetes

Платформы оркестрации контейнеров позволяют разработчикам автоматизировать процесс запуска экземпляров, подготовки хостов, связывания контейнеров для оптимизации процедур оркестрации и продления жизненных циклов приложений. Далее мы перечислим основные возможности платформы оркестрации контейнеров, потому что контейнерам нужны модули Pod, а модулям Pod нужна платформа Kubernetes для:

- предоставления доступа, не зависящего от используемой облачной технологии, ко всем возможностям сервера API;
- интеграции со всеми основными облачными платформами и гипервизорами в диспетчере контроллеров Kubernetes (Kubernetes Controller Manager, KCM);
- обеспечения отказоустойчивости для хранения и определения состояния всех сервисов, приложений и конфигураций центров обработки данных или других инфраструктур, поддерживаемых Kubernetes;
- управления развертыванием, чтобы минимизировать время простоя отдельных узлов, сервисов или приложений;
- автоматизации масштабирования хостов и приложений с поддержкой постоянного обновления;
- создания внутренних и внешних соединений (известных как типы ClusterIP, NodePort или LoadBalancer Service) с балансировкой нагрузки;
- предоставления возможности планирования запуска приложений на определенном виртуализированном оборудовании на

основе его метаданных с помощью маркировки узлов и планировщика Kubernetes;
- обеспечения высокой доступности с помощью DaemonSets и других технологических инфраструктур, в которых приоритет отдается контейнерам, работающим на всех узлах кластера;
- обнаружения сервисов через службу доменных имен (Domain Name Service, DNS), ранее реализованную как KubeDNS, а совсем недавно – CoreDNS, которая интегрируется с сервером API;
- запуска пакетных процессов (известных как задания), которые используют хранилище и контейнеры так же, как обычные приложения;
- расширения API и создания собственных программ, управляемых API, с помощью пользовательских определений ресурсов и без создания каких-либо сопоставлений портов или подключений;
- проверки сбойных процессов на уровне кластера, включая удаленное выполнение в любом контейнере в любое время с помощью `kubectl exec` и `kubectl describe`;
- подключения локального и/или удаленного хранилища к контейнеру и декларативного управления томами хранилища с помощью StorageClass API и PersistentVolumes.

На рис. 1.2 показана схема простого кластера Kubernetes. То, что делает Kubernetes, отнюдь не тривиально. Она стандартизирует управление жизненным циклом нескольких приложений, работающих в одном кластере. Основой Kubernetes является кластер, состоящий из узлов. Сложность Kubernetes – это, по общему признанию, одна из претензий, предъявляемых инженерами к Kubernetes. Сообщество усиленно работает над тем, чтобы сделать платформу проще, но вы должны понимать, что Kubernetes решает сложные задачи, которые нельзя реализовать просто с первой попытки.

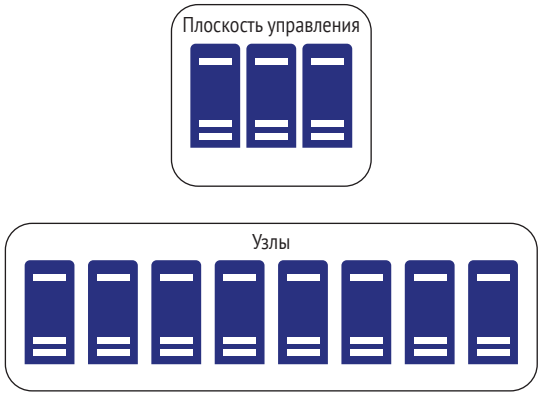

Рис. 1.2 Пример кластера Kubernetes

Если вам не нужны высокая доступность, масштабируемость и оркестрация, то, возможно, вам не нужна Kubernetes. Теперь рассмотрим типичный сценарий сбоя в кластере.

1. Узел перестает отвечать плоскости управления.
2. Плоскость управления перепланирует запуск модулей Pod, работавших на отключившемся узле, на другом узле или узлах.
3. Когда пользователь вызывает API сервера через `kubectl`, сервер сообщает об отключившемся узле и новом местоположении модулей Pod.
4. Все клиенты, взаимодействующие с сервисом в модуле Pod, переадресуются в новое местоположение.
5. Тома хранилища, подключенные к модулям Pod на неисправном узле, подключаются к новому местоположению модуля Pod, благодаря чему прежние данные остаются доступными для чтения.

Цель этой книги – дать более глубокое понимание особенностей работы основных механизмов и показать, как базовые примитивы Linux дополняют высокоуровневые компоненты Kubernetes для решения многих задач. Kubernetes опирается на сотни технологий в стеке Linux, которые часто трудно освоить и которым не хватает подробной документации. Мы надеемся, что, прочитав эту книгу, вы поймете многие тонкости Kubernetes, часто упускаемые из виду в учебных пособиях, описывающих приемы запуска контейнеров и управление ими.

Обычно Kubernetes запускается поверх неизменяемых операционных систем, когда имеется базовая ОС, обновляемая только при обновлении всей ОС (и, следовательно, является неизменной), и вы устанавливаете свои узлы/Kubernetes, используя эту ОС. Неизменяемая ОС дает много преимуществ, которые мы не будем рассматривать здесь. Вы можете запускать Kubernetes в облаке, на физических серверах или даже на Raspberry Pi. Более того, в настоящее время Министерство обороны США исследует возможность запуска Kubernetes на некоторых своих истребителях. А компания IBM реализовала поддержку запуска кластеров на своих мейнфреймах PowerPC следующего поколения.

По мере развития облачной экосистемы, окружающей Kubernetes, она будет продолжать позволять организациям находить лучшие приемы, активно вносить изменения для предотвращения проблем и поддерживать согласованность окружения, чтобы избежать дрейфа, когда некоторые машины ведут себя немного иначе, чем другие, из-за того, что на них какие-то исправления не применялись или применялись неправильно.

1.6 Компоненты и архитектура Kubernetes

Теперь рассмотрим архитектуру Kubernetes на высоком уровне (рис. 1.3). Если вкратце, то она включает ваше оборудование и ту часть вашего оборудования, на котором выполняются плоскость управления и рабочие узлы Kubernetes:

- *аппаратная инфраструктура* – включает компьютеры, сетевую инфраструктуру, инфраструктуру хранения и реестр контейнеров;
- *рабочие узлы Kubernetes* – базовая вычислительная единица в кластере Kubernetes;
- *плоскость управления Kubernetes* – основа Kubernetes. Она включает сервер API, планировщика, диспетчера контроллеров и другие контроллеры.

1.6.1 Kubernetes API

Самое важное, что можно вынести из этой главы и о чем следует помнить на протяжении чтения всей книги, – это то, что администрирование микросервисов и других контейнерных приложений на платформе Kubernetes сводится к объявлению объектов Kubernetes API. Все остальное делается автоматически самой платформой.

В этой книге мы подробно рассмотрим сервер API и его хранилище данных etcd. Почти все, что можно попросить `kubectl` сделать, сводится к чтению или записи в определенный и версионированный объект на сервере API. (Исключением являются такие операции, как использование `kubectl` для сбора журналов из действующего модуля Pod, соединение с которым проксируется через узел.) Сервер Kubernetes API – `kube-apiserver` – позволяет выполнять CRUD-операции (Create, Read, Update, Delete – создавать, читать, обновлять, удалять) со всеми объектами и предоставляет интерфейс передачи репрезентативного состояния RESTful (REpresentational State Transfer). Некоторые команды `kubectl`, такие как `describe`, возвращают комбинированное представление нескольких объектов. Как правило, все объекты Kubernetes API имеют:

- именованную версию API (например, `v1` или `rbac.authorization.k8s.io/v1`);
- тип (например, `kind: Deployment`);
- раздел метаданных.

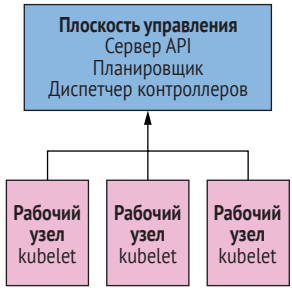

Рис. 1.3 Плоскость управления и рабочие узлы

Мы можем поблагодарить Брайана Гранта (Brian Grant), одного из первых основателей Kubernetes, за предложенную им схему управления версиями API, которая на деле доказала свою надежность. Такая

организация может показаться сложной и, честно говоря, иногда немного неудобной, но она позволяет производить обновления и устанавливать контракты, определяющие изменения в API. Изменения и миграция API часто нетривиальны, и Kubernetes предоставляет четко определенный контракт для этого. Взгляните на документы, описывающие особенности версионирования API на веб-сайте Kubernetes (http://mng.bz/voP4), где можно найти описание контрактов для версий API Alpha, Beta и GA.

Далее в этой книге мы сосредоточимся на Kubernetes, но постоянно будем возвращаться к основной теме: практически все в Kubernetes направлено на поддержку модулей Pod. В частности, мы подробно рассмотрим несколько элементов API:

- модули Pod с окружением времени выполнения и развертывания;
- детали реализации API;
- Ingress Services и балансировку нагрузки;
- хранилища PersistentVolumes и PersistentVolumeClaims;
- сетевые политики и сетевую безопасность.

Существует около 70 различных типов API, с которыми вы можете экспериментировать, создавая, изменяя и удаляя соответствующие ресурсы в стандартном кластере Kubernetes. Вы можете просмотреть их командой `kubectl api-resources`, вывод которой выглядит примерно так:

```
$ kubectl api-resources | head
NAME                     SHORTNAMES NAMESPACED KIND
bindings                            true       Binding
componentstatuses        cs         false      ComponentStatus
configmaps               cm         true       ConfigMap
endpoints                ep         true       Endpoints
events                   ev         true       Event
limitranges              limits     true       LimitRange
namespaces               ns         false      Namespace
nodes                    no         false      Node
persistentvolumeclaims   pvc        true       PersistentVolumeClaim
```

Как видите, каждый ресурс в Kubernetes API имеет:

- краткое имя;
- полное имя;
- признак ограничения пространством имен.

Пространства имен в Kubernetes позволяют объектам в файле существовать внутри определенного… эм-м… пространства имен. Это дает разработчикам простую форму иерархической группировки. Например, для приложения, запускающего 10 различных микросервисов, можно создать все его модули Pod, сервисы Service и запросы PersistentVolumeClaims (также называемые PVC) внутри одного пространства имен. При такой организации, когда придет время удалить приложение, можно просто удалить пространство имен. В главе 15 мы рассмотрим более высокоуровневые и более сложные способы ана-

лиза и организации жизненного цикла приложений. Но во многих случаях пространства имен являются наиболее очевидным и интуитивно понятным средством разделения объектов Kubernetes API, связанных с приложениями.

1.6.2 Пример первый: интернет-магазин

Представьте крупный интернет-магазин, который должен иметь возможность быстро масштабироваться в соответствии с колебаниями спроса, например, в праздничные дни. Масштабирование и прогнозирование масштабирования всегда были одной из их самых больших проблем – возможно, самой большой. Kubernetes берет на себя решение многих проблем, связанных с созданием высокодоступной и масштабируемой распределенной системы. Представьте, что у вас всегда под рукой имеются возможности масштабирования, распространения и создания высокодоступных систем. Это не только лучший способ ведения бизнеса, но и наиболее эффективная и действенная платформа для управления системами. Сочетание Kubernetes и облачных услуг позволяет задействовать чужие серверы, когда возникает потребность в дополнительных ресурсах, вместо покупки и обслуживания дополнительного оборудования на всякий случай.

1.6.3 Пример второй: онлайн-решение для благотворительности

Вторым примером из реальной жизни, о котором стоит упомянуть, может служить веб-сайт, позволяющий передавать пожертвования благотворительным организациям по выбору пользователя. Допустим, что этот конкретный сайт изначально был основан на WordPress, но с течением времени бизнес-транзакции привели к полномасштабной зависимости от фреймворков JVM (таких как Grails) с настраиваемым пользовательским интерфейсом, промежуточным уровнем и уровнем базы данных. Требования для этого бизнес-цунами включали машинное обучение, показ рекламы, обмен сообщениями, Python, Lua, NGINX, PHP, MySQL, Cassandra, Redis, Elastic, ActiveMQ, Spark, львов, тигров и медведей… да остановитесь вы уже.

Первоначальная инфраструктура была организована как созданная вручную облачная виртуальная машина, использующая Puppet для настройки всего и вся. Проект предусматривал возможность масштабирования с ростом компании, но для этого требовалось все больше и больше виртуальных машин, на которых размещались одно-два приложения. Тогда владельцы сайта решили перейти на Kubernetes. Количество виртуальных машин уменьшилось примерно с 30 до 5, и их стало легче масштабировать. Благодаря переходу на Kubernetes они полностью устранили Puppet и настройку сервера, а значит, и необходимость вручную управлять инфраструктурой.

Перейдя на Kubernetes, этой компании удалось решить целый класс проблем, связанных с администрированием виртуальных машин, нагрузкой на DNS из-за публикации сложных сервисов, и многие другие. Кроме того, время восстановления системы в случае катастрофических сбоев оказалось гораздо более предсказуемым с точки зрения инфраструктуры. Почувствовав преимущества перехода на надежную и стандартизированную методологию на основе API, способную быстро вносить массовые изменения, вы начинаете ценить декларативный характер Kubernetes и его облачный подход к оркестрации контейнеров.

1.7 Когда не стоит использовать Kubernetes

Следует признать, что в некоторых случаях Kubernetes оказывается не лучшим выбором. Вот некоторые из них:

- *высокопроизводительные вычисления (High-Performance Computing, HPC)* – контейнеры добавляют дополнительные сложности, а наличие нового уровня бьет по производительности. С развитием контейнерных технологий задержки, создаваемые контейнерами, постепенно уменьшаются, но если ваше приложение считает каждую нано- или микросекунду, то использование Kubernetes может оказаться не лучшим вариантом;
- *унаследованные приложения* – некоторые приложения имеют требования к оборудованию, программному обеспечению и задержке, что затрудняет их контейнеризацию. Например, у вас могут быть приложения, приобретенные у компании-разработчика программного обеспечения, которые официально не поддерживают работу в контейнере или в кластере Kubernetes;
- *миграция* – реализации унаследованных систем могут быть настолько жесткими, что их миграция в Kubernetes не дает особых преимуществ, кроме возможности громко заявить: «Мы используем Kubernetes». Некоторые из наиболее значительных преимуществ достигаются только после миграции, когда монолитные приложения разбиваются на логические компоненты, способные масштабироваться независимо друг от друга.

Поэтому изучайте основы и овладевайте ими. Kubernetes решает многие проблемы, описанные в этой главе, надежно и недорого.

Итоги

- Kubernetes делает жизнь проще!
- Платформа Kubernetes может работать в инфраструктуре любого типа.

- Kubernetes создает экосистему компонентов, работающих вместе. Объединение компонентов позволяет компаниям предотвращать сбои, восстанавливать и масштабировать системы в режиме реального времени, когда требуются срочные изменения.
- Все, что делается в Kubernetes, можно сделать с помощью одного простого инструмента: `kubectl`.
- Kubernetes создает кластер из одного или нескольких компьютеров и использует его как платформу для развертывания и размещения контейнеров. Kubernetes обеспечивает оркестрацию контейнеров, управление хранилищами и распределенную сеть.
- Платформа Kubernetes родилась на основе предыдущих подходов, основанных на конфигурации и контейнерах.
- Pod – это основной строительный блок Kubernetes. Поддержка модулей Pod предлагает множество возможностей: масштабирование, обработку отказов, поиск DNS и обеспечение безопасности на основе правил RBAC.
- Приложения Kubernetes управляются простыми обращениями к серверу Kubernetes API.