



# Оглавление

---

1	■ Почему появился Kubernetes.....	24
2	■ Зачем нужны модули Pod?.....	40
3	■ Создание модулей Pod.....	68
4	■ Использование контрольных групп для управления процессами в модулях Pod.....	103
5	■ Интерфейсы CNI и настройка сети в модулях Pod.....	132
6	■ Устранение проблем в крупномасштабных сетях.....	154
7	■ Хранилища в модулях Pod и CSI.....	179
8	■ Реализация и моделирование хранилищ.....	198
9	■ Запуск модулей Pod: как работает kubelet.....	221
10	■ DNS в Kubernetes.....	243
11	■ Плоскость управления.....	257
12	■ etcd и плоскость управления.....	272
13	■ Безопасность контейнеров и модулей Pod.....	296
14	■ Безопасность узлов и Kubernetes.....	312
15	■ Установка приложений.....	343

# Содержание

---

Оглавление .....	6
Предисловие .....	14
Благодарности .....	15
О книге .....	17
Об авторах .....	21
Об иллюстрации на обложке .....	23

<b>1</b>	<b>Почему появился Kubernetes .....</b>	<b>24</b>
1.1	Предварительный обзор некоторых основных терминов .....	25
1.2	Проблема дрейфа инфраструктуры и Kubernetes .....	26
1.3	Контейнеры и образы .....	27
1.4	Базовая основа Kubernetes .....	29
1.4.1	Все инфраструктурные правила в Kubernetes определяются в обычных файлах YAML .....	31
1.5	Возможности Kubernetes .....	32
1.6	Компоненты и архитектура Kubernetes .....	34
1.6.1	Kubernetes API .....	35
1.6.2	Пример первый: интернет-магазин .....	37
1.6.3	Пример второй: онлайн-решение для благотворительности .....	37
1.7	Когда не стоит использовать Kubernetes .....	38
Итоги .....	Итоги .....	38

<b>2</b>	<b>Зачем нужны модули Pod? .....</b>	<b>40</b>
2.1	Пример веб-приложения .....	42
2.1.1	Инфраструктура нашего веб-приложения .....	44
2.1.2	Эксплуатационные требования .....	45
2.2	Что такое Pod? .....	46
2.2.1	Пространства имен в Linux .....	47
2.2.2	Kubernetes, инфраструктура и Pod .....	49
2.2.3	Объект Node .....	51
2.2.4	Наше веб-приложение и плоскость управления .....	55
2.3	Создание веб-приложения с помощью kubectl .....	56
2.3.1	Сервер Kubernetes API: kube-apiserver .....	57
2.3.2	Планировщик Kubernetes: kube-scheduler .....	58

2.3.3	Контроллеры инфраструктуры .....	59
2.4	Масштабирование, высокодоступные приложения и плоскость управления .....	63
2.4.1	Автоматическое масштабирование .....	65
2.4.2	Управление затратами .....	66
Итоги	.....	67

<b>3</b>	<b>Создание модулей Pod</b> .....	68
3.1	Общий обзор примитивов Kubernetes .....	71
3.2	Что такое примитивы Linux? .....	72
3.2.1	Примитивы Linux – это инструменты управления ресурсами .....	73
3.2.2	Все сущее является файлом (или файловым дескриптором) .....	74
3.2.3	Файлы можно комбинировать .....	75
3.2.4	Настройка kind .....	76
3.3	Использование примитивов Linux в Kubernetes .....	78
3.3.1	Предварительные условия для запуска модуля Pod .....	78
3.3.2	Запуск простого модуля Pod .....	79
3.3.3	Исследование зависимостей модуля Pod от Linux .....	81
3.4	Создание модуля Pod с нуля .....	86
3.4.1	Создание изолированного процесса с помощью chroot .....	87
3.4.2	Использование mount для передачи данных процессам .....	89
3.4.3	Защита процесса с помощью unshare .....	91
3.4.4	Создание сетевого пространства имен .....	92
3.4.5	Проверка работоспособности процесса .....	93
3.4.6	Ограничение потребления процессора с помощью cgroups .....	94
3.4.7	Создание раздела resources .....	95
3.5	Использование модуля Pod в реальном мире .....	96
3.5.1	Проблема сети .....	97
3.5.2	Как kube-проху реализует сервисы Kubernetes с помощью iptables .....	98
3.5.3	Использование модуля kube-dns .....	98
3.5.4	Другие проблемы .....	100
Итоги	.....	102

<b>4</b>	<b>Использование контрольных групп для управления процессами в модулях Pod</b> .....	103
4.1	Модули Pod простаивают до завершения подготовительных операций .....	104
4.2	Процессы и потоки в Linux .....	106
4.2.1	Процессы systemd и init .....	108
4.2.2	Контрольные группы для процессов .....	109
4.2.3	Реализация контрольных групп для обычного модуля Pod .....	112
4.3	Тестирование контрольных групп .....	114
4.4	Как kubelet управляет контрольными группами .....	115
4.5	Как kubelet управляет ресурсами .....	116
4.5.1	Почему ОС не может использовать подкачку в Kubernetes? .....	117
4.5.2	Хак: настройка приоритета «для бедных» .....	118
4.5.3	Хак: настройка HugePages с помощью контейнеров инициализации .....	119
4.5.4	Классы QoS: почему они важны и как они работают .....	120
4.5.5	Создание классов QoS путем настройки ресурсов .....	121

4.6	Мониторинг ядра Linux с помощью Prometheus, cAdvisor и сервера API.....	122
4.6.1	Публикация метрик обходится недорого и имеет большую ценность .....	124
4.6.2	Почему Prometheus? .....	125
4.6.3	Создание локального сервиса мониторинга Prometheus .....	126
4.6.4	Исследование простоев в Prometheus.....	129
Итоги	.....	131

## 5 Интерфейсы CNI и настройка сети в модулях Pod..... 132

5.1	Зачем нужны программно-определяемые сети в Kubernetes .....	134
5.2	Реализация Kubernetes SDN на стороне сервиса: kube-proxy.....	136
5.2.1	Плоскость данных в kube-proxy .....	138
5.2.2	Подробнее о NodePort .....	140
5.3	Провайдеры CNI .....	141
5.4	Два плагина CNI: Calico и Antrea .....	143
5.4.1	Архитектура плагинов CNI .....	143
5.4.2	Давайте поэкспериментируем с некоторыми CNI .....	144
5.4.3	Установка провайдера CNI Calico .....	146
5.4.4	Организация сети в Kubernetes с OVS и Antrea.....	149
5.4.5	Замечание о провайдерах CNI и kube-proxy в разных ОС .....	152
Итоги	.....	153

## 6 Устранение проблем в крупномасштабных сетях..... 154

6.1	Sonobuou: инструмент подтверждения работоспособности кластера.....	155
6.1.1	Трассировка движения данных модулей Pod в кластере .....	156
6.1.2	Настройка кластера с CNI-провайдером Antrea .....	157
6.2	Исследование особенностей маршрутизации в разных провайдерах CNI с помощью команд <code>arp</code> и <code>ip</code> .....	158
6.2.1	Что такое IP-туннель и почему его используют провайдеры CNI? .....	159
6.2.2	Сколько пакетов проходит через сетевые интерфейсы CNI?.....	160
6.2.3	Маршруты .....	161
6.2.4	Инструменты для CNI: Open vSwitch (OVS).....	163
6.2.5	Трассировка движения данных активных контейнеров с помощью <code>tcpdump</code> .....	164
6.3	kube-proxy и iptables.....	166
6.3.1	iptables-save и diff.....	166
6.3.2	Как сетевые политики изменяют правила CNI .....	167
6.3.3	Как реализуются политики?.....	170
6.4	Входные контроллеры.....	172
6.4.1	Настройка Contour и кластера kind для изучения входных контроллеров .....	173
6.4.2	Настройка простого модуля Pod с веб-сервером .....	174
Итоги	.....	178

<b>7</b>	<b>Хранилища в модулях Pod и CSI</b> .....	179
7.1	Небольшое отступление: виртуальная файловая система (VFS) в Linux .....	181
7.2	Три вида хранилищ для Kubernetes .....	182
7.3	Создание PVC в кластере kind.....	184
7.4	Интерфейс контейнерного хранилища (CSI) .....	188
7.4.1	Проблема внутреннего провайдера .....	189
7.4.2	CSI как спецификация, работающая внутри Kubernetes .....	191
7.4.3	CSI: как работает драйвер хранилища .....	193
7.4.4	Привязка точек монтирования.....	193
7.5	Краткий обзор действующих драйверов CSI.....	194
7.5.1	Контроллер .....	194
7.5.2	Интерфейс узла .....	195
7.5.3	CSI в операционных системах, отличных от Linux .....	196
	Итоги .....	196
<b>8</b>	<b>Реализация и моделирование хранилищ</b> .....	198
8.1	Микрокосм в экосистеме Kubernetes: динамическое хранилище ....	199
8.1.1	Оперативное управление хранилищем: динамическое выделение ресурсов .....	200
8.1.2	Локальное хранилище в сравнении с emptyDir .....	201
8.1.3	Тома PersistentVolume .....	203
8.1.4	Интерфейс контейнерного хранилища (CSI) .....	204
8.2	Динамическая подготовка выигрывает от CSI, но не зависит от него .....	205
8.2.1	Классы хранилищ (StorageClass) .....	206
8.2.2	Вернемся к центрам обработки данных.....	207
8.3	Варианты организации хранилищ в Kubernetes .....	209
8.3.1	Секреты: эфемерная передача файлов .....	209
8.4	Как выглядит типичный провайдер динамического хранилища? ....	212
8.5	hostPath для управления системой и/или доступа к данным .....	214
8.5.1	hostPath, CSI и CNI: канонический вариант использования .....	214
8.5.2	Cassandra: пример реального хранилища в Kubernetes .....	217
8.5.3	Дополнительные возможности и модель хранения в Kubernetes.....	218
8.6	Дополнительная литература.....	219
	Итоги .....	220
<b>9</b>	<b>Запуск модулей Pod: как работает kubelet</b> .....	221
9.1	kubelet и узел .....	222
9.2	Основы kubelet.....	223
9.2.1	Среда выполнения контейнеров: стандарты и соглашения .....	224
9.2.2	Конфигурационные параметры и API агента kubelet.....	225
9.3	Создание модуля Pod и его мониторинг.....	228
9.3.1	Запуск kubelet .....	229
9.3.2	После запуска: жизненный цикл узла .....	230
9.3.3	Механизм аренды и блокировки в etcd, эволюция аренды узла .....	230
9.3.4	Управление жизненным циклом Pod в kubelet.....	231
9.3.5	CRI, контейнеры и образы: как они связаны .....	233
9.3.6	kubelet не запускает контейнеры: это делает CRI.....	233
9.3.7	Приостановленный контейнер: момент истины .....	235

9.4	Интерфейс времени выполнения контейнеров (CRI).....	235
9.4.1	Сообщаем Kubernetes, где находится среда выполнения контейнеров .....	235
9.4.2	Процедуры CRI .....	236
9.4.3	Абстракция kubelet вокруг CRI: GenericRuntimeManager .....	236
9.4.4	Как вызывается CRI? .....	237
9.5	Интерфейсы kubelet .....	237
9.5.1	Внутренний интерфейс среды выполнения .....	237
9.5.2	Как kubelet извлекает образы: интерфейс ImageService .....	239
9.5.3	Передача ImagePullSecret в kubelet .....	240
9.6	Дополнительная литература.....	241
Итоги	.....	241

<b>10</b>	<b>DNS в Kubernetes</b> .....	243
10.1	Краткое введение в DNS (и CoreDNS).....	243
10.1.1	NXDOMAIN, записи A и записи CNAME .....	244
10.1.2	Модулям Pod нужен внутренний DNS .....	246
10.2	Почему StatefulSet, а не Deployment?.....	248
10.2.1	DNS и автономные сервисы .....	248
10.2.2	Постоянные записи DNS в StatefulSet .....	250
10.2.3	Развертывание с несколькими пространствами имен для изучения свойств модуля DNS .....	250
10.3	Файл resolv.conf .....	252
10.3.1	Краткое примечание о маршрутизации .....	252
10.3.2	CoreDNS: вышестоящий сервер имен для ClusterFirst DNS.....	254
10.3.3	Разбор конфигурации плагина CoreDNS .....	255
Итоги	.....	256

<b>11</b>	<b>Плоскость управления</b> .....	257
11.1	Плоскость управления .....	258
11.2	Особенности сервера API .....	259
11.2.1	Объекты API и пользовательские ресурсы .....	259
11.2.2	Определения пользовательских ресурсов (CRD) .....	261
11.2.3	Планировщик .....	261
11.2.4	Краткий обзор фреймворка планирования .....	267
11.3	Диспетчер контроллеров .....	267
11.3.1	Хранилище .....	268
11.3.2	Учетные данные сервисов и токены .....	269
11.4	Облачные диспетчеры контроллеров Kubernetes (CCM) .....	269
11.5	Дополнительная литература.....	271
Итоги	.....	271

<b>12</b>	<b>etcd и плоскость управления</b> .....	272
12.1	Заметки для нетерпеливых .....	273
12.1.1	Мониторинг производительности etcd с помощью Prometheus .....	274
12.1.2	Когда нужно настраивать etcd .....	278
12.1.3	Пример: быстрая проверка работоспособности etcd .....	280
12.1.4	etcd v3 и v2 .....	280
12.2	etcd как хранилище данных .....	281
12.2.1	Можно ли запустить Kubernetes в других базах данных? .....	281

12.2.2	Строгая согласованность .....	283
12.2.3	Согласованность в etcd обеспечивают операции fsync .....	283
12.3	Обзор интерфейса Kubernetes с etcd .....	285
12.4	Задача etcd – надежное хранение фактов.....	285
12.4.1	Журнал упреждающей записи etcd .....	287
12.4.2	Влияние на Kubernetes .....	287
12.5	Теорема CAP.....	287
12.6	Балансировка нагрузки на уровне клиента и etcd .....	289
12.6.1	Ограничения по размеру: о чем (не) следует беспокоиться .....	289
12.7	Шифрование хранимых данных в etcd .....	291
12.8	Производительность и отказоустойчивость etcd в глобальном масштабе .....	292
12.9	Интервал отправки контрольных сообщений в высокораспределенной etcd .....	292
12.10	Настройка клиента etcd в кластере kind .....	293
12.10.1	Запуск etcd в окружении, отличном от Linux.....	294
Итоги	.....	295

## 13 Безопасность контейнеров и модулей Pod..... 296

13.1	Радиус взрыва .....	297
13.1.1	Уязвимости .....	298
13.1.2	Вторжение .....	298
13.2	Безопасность контейнера .....	298
13.2.1	Планирование обновления контейнеров и пользовательского программного обеспечения .....	299
13.2.2	Контроль контейнеров .....	299
13.2.3	Пользователи в контейнерах – не запускайте ПО от имени root .....	300
13.2.4	Используйте наименьшие возможные контейнеры.....	300
13.2.5	Происхождение контейнера .....	301
13.2.6	Линтеры для контейнеров .....	302
13.3	Безопасность модулей Pod.....	302
13.3.1	Контекст безопасности .....	303
13.3.2	Расширение привилегий и возможностей .....	305
13.3.3	Политики безопасности Pod (PSP) .....	307
13.3.4	Не внедряйте автоматически токен учетной записи сервиса .....	309
13.3.5	Модули Pod с привилегиями root .....	309
13.3.6	Граница безопасности .....	310
Итоги	.....	311

## 14 Безопасность узлов и Kubernetes .....

14.1	Безопасность узла.....	312
14.1.1	Сертификаты TLS.....	313
14.1.2	Неизменяемые ОС и применение исправлений на узлах .....	314
14.1.3	Изолированные среды выполнения контейнеров .....	315
14.1.4	Атаки на ресурсы .....	316
14.1.5	Единицы измерения потребления процессора .....	317
14.1.6	Единицы измерения объема памяти .....	317
14.1.7	Единицы измерения объема хранилища .....	318
14.1.8	Сети хостов и модулей Pod .....	318
14.1.9	Пример модуля Pod .....	319

14.2	Безопасность сервера API .....	320
14.2.1	Управление доступом на основе ролей (RBAC) .....	320
14.2.2	Определение RBAC API .....	321
14.2.3	Ресурсы и подресурсы .....	323
14.2.4	Субъекты и RBAC.....	325
14.2.5	Отладка RBAC.....	326
14.3	Authn, Authz и Secret .....	326
14.3.1	Учетные записи сервисов IAM: защита облачных API .....	327
14.3.2	Доступ к облачным ресурсам .....	328
14.3.3	Частные серверы API .....	329
14.4	Безопасность сети .....	329
14.4.1	Сетевые политики.....	330
14.4.2	Балансировщики нагрузки .....	334
14.4.3	Агент открытой политики (OPA) .....	335
14.4.4	Коллективная аренда.....	338
14.5	Советы по Kubernetes .....	341
Итоги	.....	341

## 15 Установка приложений .....

15.1	Размышления о приложениях в Kubernetes .....	344
15.1.1	Масштаб приложения влияет на выбор инструментов .....	345
15.2	Приложения на основе микросервисов обычно требуют тысячи строк определения конфигурации .....	345
15.3	Переосмысление установки приложения Guestbook в реальных условиях .....	346
15.4	Установка набора инструментов Carvel.....	347
15.4.1	Часть 1: разделение ресурсов на отдельные файлы .....	347
15.4.2	Часть 2: исправление файлов приложения с помощью ytt.....	349
15.4.3	Часть 3: развертывание приложения Guestbook и управление им .....	351
15.4.4	Часть 4: создание оператора karr для упаковки приложения и управления им.....	355
15.5	И снова об операторах Kubernetes .....	359
15.6	Tanzu Community Edition: пример комплексного набора инструментов Carvel .....	362
Итоги	.....	363

Предметный указатель.....	365
---------------------------	-----

# 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-проху), 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 Cilium;
- 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](http://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](mailto: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/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу [dmkpress@gmail.com](mailto: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](mailto: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](mailto: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 Джейем 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.

# 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.

Преыдушие 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 требовалась 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.

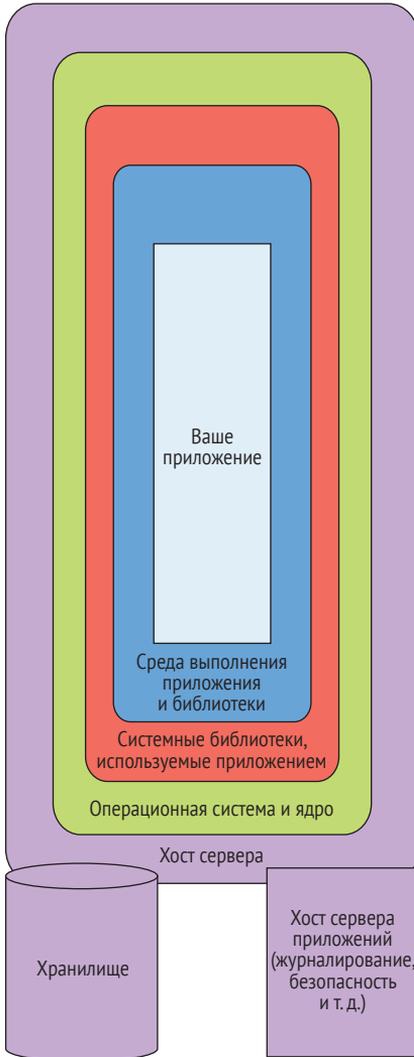


Рис. 1.1 Приложения, выполняющиеся в контейнерах

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 немислимо 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 ОСИ 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/mysql"]
```

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 ОС 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.

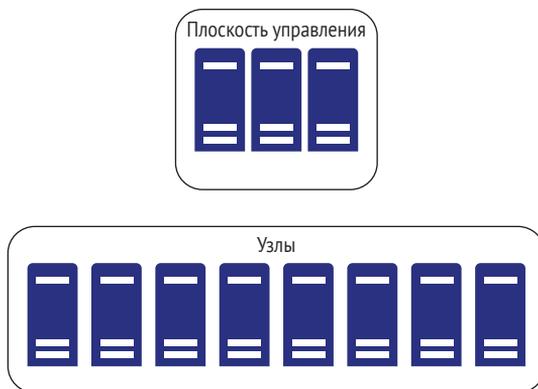


Рис. 1.2 Пример кластера Kubernetes

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.

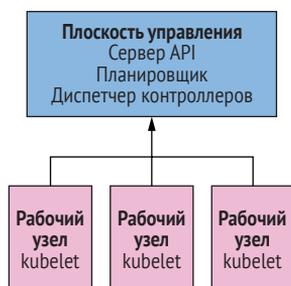


Рис. 1.3 Плоскость управления и рабочие узлы

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          SHORTRNAMES  NAMESPACED  KIND
bindings      cs            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 овладейте 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.