

# Содержание

<b>От издательства</b> .....	13
<b>Об авторе</b> .....	14
<b>О рецензенте</b> .....	15
<b>Предисловие</b> .....	16
<b>Часть I. ОСНОВЫ DOCKER COMPOSE</b> .....	19
<b>Глава 1. Введение в Docker Compose</b> .....	20
Технические требования.....	21
Знакомство с Docker Compose и особенностями его использования.....	21
Установка Docker Compose.....	21
Docker Desktop.....	22
Установка Docker.....	22
В macOS.....	22
В Windows .....	23
В Linux .....	25
docker compose и docker-compose.....	27
Знакомство с принципами работы Docker Compose .....	27
Ваш первый файл Docker Compose.....	29
Использование образа Docker в Docker Compose .....	32
Итоги.....	35
<b>Глава 2. Запуск первого приложения с помощью Compose</b> .....	36
Технические требования.....	37
Создание простого приложения .....	37
Установка Go.....	37
REST API в Go с использованием Gin .....	37

Приложение .....	38
Запуск Redis с помощью Compose .....	40
Использование командной оболочки в контейнере, управляемом Compose .....	41
Взаимодействие со службой Docker Compose .....	42
Упаковка приложения с помощью Docker и Compose .....	44
Передача конфигурации через окружение .....	45
Создание образа Docker .....	46
Запуск образа .....	46
Создание образа с помощью Compose .....	48
Сборка и определение имени образа .....	48
Запуск многоконтейнерного приложения с помощью Compose .....	49
Проверка работоспособности .....	49
Как действует проверка работоспособности .....	49
Добавление проверки работоспособности в Compose .....	50
Зависимости от служб .....	51
Точка входа, аргументы и переменные окружения .....	52
Файлы с переменными окружения .....	52
Сценарий .....	52
Настройка точки входа .....	53
Метки .....	53
Образы .....	54
Контейнеры .....	54
Итоги .....	55
<b>Глава 3. Основы сетей и томов .....</b>	<b>56</b>
Технические требования .....	57
Основы томов Docker .....	57
Подключение тома Docker к контейнеру .....	57
Общие тома .....	59
Тома только для чтения .....	59
Драйверы томов Docker .....	60
Использование драйвера тома вместо локального монтирования .....	61
Объявление томов Docker в файлах Compose .....	61
Подключение томов Docker к существующему приложению .....	62
Создание конфигурационного файла .....	63
Монтирование файла с использованием тома .....	63
Монтирование томов только для чтения .....	64
Основы сетей Docker .....	66
Мостовая сеть .....	67
Мостовая сеть, определяемая пользователем .....	67
Сеть хоста .....	68
Оверлейная сеть .....	68
Определение сетей в конфигурации Compose .....	68
Добавление дополнительной сети в текущее приложение .....	70
Итоги .....	72

<b>Глава 4. Выполнение команд Docker Compose</b> .....	73
Технические требования.....	73
Введение в команды Compose.....	74
Интерфейс командной строки Docker и команды Compose.....	74
Настройка целевого приложения.....	74
Команды подготовки.....	75
build.....	75
create.....	76
up.....	76
Команды управления контейнерами.....	78
exec.....	78
run.....	78
pause.....	79
unpause.....	79
start и stop.....	80
restart.....	80
kill.....	81
ps.....	81
Команды освобождения ресурсов.....	82
down.....	82
rm.....	84
Команды управления образами.....	85
Список образов.....	86
Извлечение образов.....	86
Отправка образов.....	87
Локальный реестр Docker в Compose.....	87
Отправка в локальный реестр.....	88
Команды мониторинга.....	89
logs.....	89
top.....	90
events.....	90
Другие команды.....	91
help.....	91
version.....	91
port.....	91
config.....	91
Итоги.....	92

## **Часть II. ПОВСЕДНЕВНАЯ РАЗРАБОТКА С ПОМОЩЬЮ DOCKER COMPOSE**..... 93

### **Глава 5. Подключение микросервисов друг к другу**..... 94

Технические требования.....	95
Микросервис определения местоположения.....	95

Добавление службы определения местоположения в Compose .....	101
Добавление сети для микросервиса местоположения .....	102
Выполнение запросов к микросервису местоположения.....	103
Потоковая передача событий задач.....	105
Добавление микросервиса обработки событий задач .....	107
Итоги.....	109
<b>Глава 6. Службы мониторинга с Prometheus .....</b>	<b>110</b>
Что такое Prometheus?.....	110
Добавление конечной точки для Prometheus .....	111
Добавление конечной точки метрик в диспетчер задач .....	111
Добавление конечной точки метрик в службу определения местоположения .....	112
Экспорт метрик из службы событий.....	113
Настройка анализа метрик в Prometheus.....	114
Добавление Prometheus в сеть Compose.....	116
Отправка метрик в Prometheus .....	118
Первый запрос метрик.....	119
Добавление уведомления .....	120
Итоги.....	122
<b>Глава 7. Комбинирование файлов Compose .....</b>	<b>123</b>
Технические требования.....	124
Разделение файлов Compose.....	124
Основа для диспетчера задач .....	124
Служба определения местоположения.....	125
Служба событий .....	126
Диспетчер задач.....	126
Prometheus .....	127
Объединение файлов Compose .....	128
Выбор файлов Compose для запуска .....	129
Использование Hoverfly .....	129
Наследование служб .....	129
Захват трафика с помощью Hoverfly .....	130
Извлечение данных для имитации службы определения местоположения .....	132
Извлечение данных для имитации службы Pushgateway.....	132
Адаптация моделирования.....	132
Создание фиктивных приложений с помощью Hoverfly .....	133
Имитация службы определения местоположения .....	133
Имитация Pushgateway.....	133
Создание различных окружений.....	134
Запуск с включенным захватом .....	134

Запуск с отключенным мониторингом .....	135
Запуск приложений по отдельности .....	135
Объединение нескольких файлов Compose в один .....	136
Использование команды <code>config</code> .....	136
Итоги .....	137

## **Глава 8. Локальное моделирование промышленного окружения .....**

Технические требования .....	138
Разделение приватных и общедоступных рабочих нагрузок .....	139
Настройка локальной службы DynamoDB .....	140
Создание таблиц DynamoDB .....	140
Взаимодействие с локальной DynamoDB .....	141
Настройка локальной службы SQS .....	142
Настройка локальной службы S3 .....	143
Настройка функции Lambda на основе REST .....	144
Настройка функции Lambda на основе SQS .....	147
Ссылки Docker Compose .....	148
Соединение функций Lambda .....	149
Итоги .....	151

## **Глава 9. Создание расширенных заданий CI/CD .....**

Технические требования .....	154
Введение в CI/CD .....	154
Использование действий GitHub Actions с Docker Compose .....	155
Создание первого действия GitHub Action .....	155
Кеширование собранных образов .....	156
Создание образов приложений .....	157
Тестирование приложения Compose .....	157
Использование конвейеров Bitbucket с Docker Compose .....	158
Создание первого конвейера Bitbucket .....	158
Кеширование образов Compose и Docker .....	159
Создание образов приложений .....	160
Тестирование приложения Compose .....	161
Использование Travis с Docker Compose .....	162
Создание первого задания в Travis .....	162
Кеширование Compose .....	162
Создание образов приложений .....	163
Тестирование приложения Compose .....	163
Итоги .....	164

<b>Часть III. РАЗВЕРТЫВАНИЕ С ПОМОЩЬЮ DOCKER COMPOSE</b> .....	165
<b>Глава 10. Развертывание Docker Compose на удаленных хостах</b> .....	166
Технические требования.....	167
Удаленные хосты Docker .....	167
Создание удаленного хоста Docker.....	167
Создание хоста Docker в AWS EC2 .....	167
Установка Terraform .....	168
Настройка машины EC2 с включенным SSH .....	169
Использование удаленного хоста Docker .....	172
Контексты Docker .....	173
Развертывание Compose на удаленных хостах .....	174
Удаленное развертывание хоста через IDE.....	175
Итоги.....	176
<b>Глава 11. Развертывание Docker Compose в AWS</b> .....	177
Технические требования.....	178
Введение в AWS ECS.....	178
Размещение образов Docker в AWS ECR .....	179
Подготовка ECR с помощью AWS CLI.....	179
Создание ECR с помощью Terraform.....	180
Хранение файла состояния Terraform .....	181
Отправка образов в ECR.....	182
Адаптация образов приложения Compose.....	183
Развертывание приложения в кластере ECS.....	184
Запуск приложения Compose в существующем кластере .....	187
Создание группы журналов .....	187
Создание приватной сети.....	188
Группы безопасности .....	190
Настройка кластера ECS и балансировщика нагрузки.....	190
Обновление файла Compose .....	190
Запуск приложения Compose в существующей инфраструктуре .....	191
Расширенные концепции Docker Compose в ECS.....	192
Обновление приложения.....	192
Масштабирование приложения .....	193
Использование секретов.....	194
Итоги.....	195

<b>Глава 12. Развертывание Docker Compose в Azure</b> .....	196
Технические требования.....	196
Введение в ACI.....	197
Отправка контейнеров в реестр Azure.....	197
Сохранение файла состояния Terraform.....	199
Развертывание в ACI.....	200
Итоги.....	204
<b>Глава 13. Миграция на конфигурацию Kubernetes с помощью Compose</b> .....	205
Технические требования.....	206
Введение в Kubernetes.....	206
Компоненты Kubernetes и Compose .....	207
Приложения Compose и пространства имен .....	207
Службы Compose и службы Kubernetes.....	207
Метки.....	208
Сети Compose и сетевые политики Kubernetes .....	208
Преобразование файлов с помощью Compose .....	208
Введение в Minikube .....	210
Развертывание в Kubernetes.....	212
Итоги.....	214
<b>Предметный указатель</b> .....	215

# Об авторе

**Эммануил Гадзурас (Emmanouil Gkatzouras)** начал свой путь в области разработки программного обеспечения, когда поступил на работу в департамент вычислительной техники и информатики в городе Патры (Греция). Затем он работал инженером-программистом в различных компаниях. В 2015 году поступил на работу в Oseven, где начал работать с облачными провайдерами, такими как AWS и Azure, и инструментами оркестрации контейнеров, такими как ECS и Kubernetes. Он занимал самые разные должности и в настоящее время работает как облачный архитектор в команде платформы.

Он любит помогать сообществу разработчиков, активно участвуя в проектах с открытым исходным кодом, таких как InfluxDB, Spring Cloud GCP и Alpacka, а также публикуя в блоге статьи по информатике на различные темы. Старается непрерывно учиться и уже обладает многими сертификатами, такими как CKA, CCDAK, PSM, CKAD и PSO.

*Хочу поблагодарить самого себя за то, что собрался с силами и нашел время для написания этой книги, не прекращая при этом выполнять свои повседневные обязанности. Также хочу поблагодарить мою подругу Вив (Viv) за то, что терпела, пока я писал эту книгу, и всю команду редакторов из издательства Packt, которые помогли мне: Роми Диас (Romy Dias), Ашвин Динеш Харва (Ashwin Dinesh Kharwa) и Ниранджан Найквади (Niranjan Naikwadi).*



# О рецензенте

**Вернер Дейкерман (Werner Dijkerman)** – специалист в облачных технологиях, Kubernetes (сертифицированный специалист) и DevOps. В настоящее время занимается облачными решениями и инструментами, включая AWS, Ansible, Kubernetes и Terraform, а также платформами IaaS (Infrastructure as a Code – инфраструктура как код). Широко использует инструменты мониторинга и автоматизации, такие как Zabbix, Prometheus и ELK Stack, и старается автоматизировать все и вся, чтобы максимально исключить ручную работу.

*Большое спасибо, обнимаю и привет  
Эрнсту Форстефельду (Ernst Vorsteveld)!*

# Предисловие

В этой книге описываются основы Docker Compose и демонстрируется его практическое применение. Сначала вы узнаете, какие компоненты входят в состав Docker Compose, какие команды доступны, их назначение и как они используются. Затем мы рассмотрим настройку баз данных, приемы использования сетевых возможностей Docker и организации взаимодействий между микросервисами. Вы также узнаете, как запускать целые стеки локально в Compose, как моделировать промышленные окружения и расширять задания CI/CD с помощью средств, предлагаемых Docker Compose. Наконец, вы познакомитесь с дополнительными темами, такими как использование Docker Compose для развертывания промышленных окружений, подготовки инфраструктуры в общедоступных облаках, таких как AWS и Azure, а также проложите путь для миграции на механизм оркестрации Kubernetes.

## Кому адресована эта книга

Эта книга адресована инженерам-программистам, разработчикам и инженерам DevOps, которые желают научиться настраивать многоконтейнерные приложения Docker с помощью Compose без необходимости изучать и настраивать механизм оркестрации Docker. Она также будет полезна руководителям групп, стремящимся повысить продуктивность команд разработчиков за счет оптимизации подготовки сложных окружений разработки с помощью Docker Compose.

## О чем рассказывается в книге

*Глава 1 «Введение в Docker Compose»* перечисляет различные особенности и варианты использования Compose. Дает краткое описание формата файла Docker Compose и приводит первый работающий пример на основе Compose.

*Глава 2 «Запуск первого приложения с помощью Compose»* показывает, как создать простое приложение Golang, взаимодействующее с базой данных Redis. К концу вы научитесь запускать многоконтейнерные приложения с помощью Compose.

*Глава 3 «Основы сетей и томов»* посвящена основам организации томов и сетей в Docker. К концу вы научитесь настраивать и использовать сеть для существующего приложения.

*Глава 4 «Выполнение команд Docker Compose»* знакомит с командами Compose, их назначением и вариантами использования.

*Глава 5 «Подключение микросервисов друг к другу»* рассматривает особенности создания новых микросервисов. К концу вы научитесь разрабатывать новые микросервисы, работающие в одной сети, и связывать их друг с другом.

*Глава 6 «Мониторинг служб с помощью Prometheus»* рассказывает, как организовать мониторинг служб с помощью Prometheus.

*Глава 7 «Комбинирование файлов Compose»* рассматривает вопросы модульной организации файлов Compose и разделение их на несколько частей.

*Глава 8 «Локальное моделирование промышленного окружения»* дает обзор сложных конфигураций Compose с целью частичного или полного моделирования промышленного окружения в локальной среде.

*Глава 9 «Создание расширенных заданий CI/CD»* показывает, как создавать сложные задачи CI/CD путем моделирования с помощью Compose.

*Глава 10 «Развертывание Docker Compose на удаленных хостах»* описывает приемы развертывания на удаленных хостах с помощью Compose.

*Глава 11 «Развертывание Docker Compose в AWS»* демонстрирует применение имеющихся знаний о Compose для развертывания в AWS с использованием ECS.

*Глава 12 «Развертывание Docker Compose в Azure»* посвящена еще одному популярному облачному провайдеру – Azure. К концу вы научитесь выполнять развертывание в Azure ACI.

*Глава 13 «Миграция на конфигурацию Kubernetes с помощью Compose»* показывает, как преобразовать файлы Compose в развертывание Kubernetes.

## КАК ПОЛУЧИТЬ МАКСИМУМ ОТ ЭТОЙ КНИГИ

Предполагается, что читатель понимает основные идеи контейнеризации и имеет базовые знания о Docker. Также желательно иметь навыки работы в командной строке. Лучшим вариантом для изучения книги было бы параллельное использование рабочей станции с UNIX для опробования примеров. Большая часть представленного кода и команд также должна работать на компьютерах с Windows.

Если вы читаете электронную версию этой книги, то мы советуем вводить код самостоятельно или получить его из репозитория книги на GitHub (ссылка приводится в следующем разделе). Это поможет вам избежать возможных ошибок, связанных с копированием и вставкой кода.

## ТИПОГРАФСКИЕ СОГЛАШЕНИЯ

В этой книге используется ряд соглашений по оформлению текста.

Код в тексте: так оформляются фрагменты программного кода в тексте, имена таблиц баз данных, имена папок, имена файлов, расширения файлов,

пути к файлам, фиктивные URL, ввод пользователя и ссылки Twitter. Например: «Смонтируйте созданный файл конфигурации `nginx.conf` как любой другой файл в вашей системе».

Блоки кода оформляются так:

```
type Task struct {
    Id          string `json:"id"`
    Name        string `json:"name"`
    Description string `json:"description"`
    Timestamp   int64  `json:"timestamp"`
}
```

Фрагменты кода, заслуживающие особого внимания, будут выделяться жирным шрифтом:

```
services:
  redis:
  image: redis
  ports:
  - 6379:6379
```

Любой ввод или вывод в командной строке будет оформляться так:

```
$ curl --location --request POST 'localhost:8080/task/'
$ cat /etc/nginx/nginx.conf
```

**Жирным шрифтом** будут выделяться новые термины, важные определения или слова, которые видны на экране. Например, надписи в меню или в диалоговых окнах будут выделены жирным шрифтом. Например: «Выберите раздел **System info** (Информация о системе) в панели администратора».



Советы или важные примечания будут оформляться так.

Часть I

---

# ОСНОВЫ DOCKER COMPOSE

Эта часть знакомит с Docker Compose и его работой за кулисами. Мы познакомимся с Compose, разработав и развернув с его помощью набор приложений. Мы также узнаем, как идеи Docker, используемые ежедневно (такие как сети и тома), отражаются в Compose. Наконец, мы перечислим доступные команды Compose и посмотрим, как они выполняются.

Данная часть включает следующие главы:

- главу 1 «Введение в Docker Compose»;
- главу 2 «Запуск первого приложения с помощью Compose»;
- главу 3 «Основы сетей и томов»;
- главу 4 «Выполнение команд Docker Compose».

# Глава 1

## Введение в Docker Compose

**Docker** быстро стал неотъемлемой частью разработки и развертывания приложений, поэтому вам часто придется сталкиваться с инструментом **Docker Compose**. Возможно, вы читали о нем, использовали его или даже сталкивались с ним, просматривая официальную документацию Docker.

С развитием программного обеспечения для приложений стало обычным делом взаимодействовать с несколькими программными компонентами. Популярные приложения часто сталкиваются с необходимостью разделения рабочих нагрузок и масштабирования. Разделение логики и ответственности между несколькими программными компонентами неизбежно. Docker предлагает решения, упрощающие контейнеризацию и изоляцию рабочих нагрузок, а также управление ими. А Docker Compose может помочь в разработке современных многоконтейнерных приложений и их развертывании.

Docker Compose – простой и эффективный инструмент. Его применение помогает решать проблемы, возникающие при работе с многоконтейнерными приложениями, и повышать продуктивность разработчиков. Docker Compose можно использовать не только в жизненном цикле разработки, но также в промышленных окружениях, что позволяет устранить разрыв между разработкой в локальной среде и фактическими развертываниями в промышленных окружениях. Эту возможность можно использовать для плавного перехода к механизмам оркестрации, таким как Kubernetes.

В данной главе мы кратко познакомимся с основными возможностями Compose, принципами его работы и типичными вариантами использования. Мы установим Docker Compose и создадим первый файл Compose для запуска программного компонента. Исследуя формат файла Compose, мы также применим некоторые дополнительные настройки и используем один из наших локальных образов.

В этой главе рассматриваются следующие темы:

- знакомство с Docker Compose и особенностями его использования;
- установка Docker Compose;
- знакомство с принципами работы Docker Compose;

- создание первого файла Docker Compose;
- использование образа Docker в Docker Compose.

## ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Примеры кода для этой книги можно найти в репозитории GitHub по адресу <https://github.com/PacktPublishing/A-Developer-s-Essential-Guide-to-Docker-Compose>. Если мы будем обновлять код, то все обновления будут доступны в этом репозитории.

## ЗНАКОМСТВО С DOCKER COMPOSE И ОСОБЕННОСТЯМИ ЕГО ИСПОЛЬЗОВАНИЯ

Docker Compose – это инструмент для определения и запуска многоконтейнерных приложений Docker. Конфигурация определяется с помощью файлов YAML, а с помощью **утилиты командной строки Docker Compose** можно определять и выполнять операции с контейнерами, управляемыми Docker Compose.

Вот список функций, которые предлагает Compose:

- запуск сложных многоконтейнерных приложений на одном хосте;
- изоляция рабочих нагрузок Docker;
- загрузка и распределение сложных приложений;
- создание нескольких окружений;
- сохранение данных об изменении приложения;
- обновление версий приложений;
- конструирование окружения;
- повторное использование конфигураций;
- моделирование сложных промышленных окружений;
- развертывание промышленных приложений.

В этой книге мы подробно рассмотрим все эти функции, увидим, какую пользу можно извлечь из них, и добавим их в наш процесс разработки. В следующем разделе мы установим Docker и Compose на рабочую станцию, используя выбранную вами операционную систему.

## УСТАНОВКА DOCKER COMPOSE

Docker Compose и Compose CLI написаны на языке Go. Compose поддерживает три основные операционные системы: Linux, Windows и macOS. Compose предназначен для управления многоконтейнерными приложениями Docker,

поэтому наличие установленного программного обеспечения Docker является обязательным предварительным условием.

## Docker Desktop

В Mac и Windows **Docker Desktop** – это вариант установки Docker Compose. Docker Desktop упрощает настройку Docker на локальном компьютере. Он создает **виртуальную машину Linux** на хосте и поддерживает взаимодействие контейнеров с ОС, например доступ к файловой системе и сети. Установка Docker Desktop **выполняется** в один щелчок и добавляет в систему все необходимые инструменты, такие как Docker CLI. Одним из этих инструментов является Docker Compose. Поэтому установки Docker Desktop достаточно для взаимодействия с Docker Engine с помощью Compose.

## Установка Docker

Чтобы установить правильный дистрибутив Docker для выбранной рабочей станции, перейдите в соответствующий раздел на официальной странице Docker:

- Docker Desktop для Mac: <https://docs.docker.com/desktop/mac/install/>;
- Docker Desktop для Windows: <https://docs.docker.com/desktop/windows/install/>;
- Docker Engine для Linux: <https://docs.docker.com/engine/install/>.

### *В macOS*

Apple поставляет рабочие станции с двумя типами процессоров: процессором Intel и процессором Apple. Docker имеет дистрибутивы для обоих. После завершения загрузки, щелкнув на установщике, вы сможете перетащить приложение Docker, как показано на следующем скриншоте (рис. 1.1).



Рис. 1.1 ❖ Установка Docker в Mac



По завершении установки можно проверить работоспособность Docker, выполнив команду `hello world`:

```
$ docker run --rm hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
93288797bd35: Pull complete
Digest: sha256:97a379f4f88575512824f3b352bc03cd75e239179eea
0fecc38e597b2209f49a
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working
correctly.
..
```

Дополнительно можно проверить работоспособность Compose:

```
$ docker compose version
Docker Compose version v2.2.3
```

Теперь посмотрим, как установить Docker Desktop в Windows.

## В Windows

В Windows, как и в Mac, установка Docker Desktop не вызывает никаких сложностей.

После загрузки и запуска установочного файла EXE в систему будет установлен Docker вместе со всеми утилитами. Затем необходимо выполнить некоторые дополнительные настройки, чтобы включить виртуализацию в Windows.

Независимо от используемого механизма – WSL 2 или Hyper-V – необходимо настроить BIOS компьютера и включить в нем виртуализацию, как показано на следующем скриншоте (рис. 1.2).

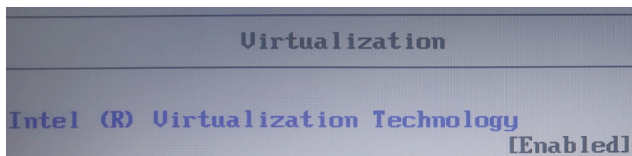


Рис. 1.2 ❖ Включение виртуализации в BIOS

После входа в Windows включите соответствующие функции виртуализации.

Для WSL 2 необходимо включить функцию **Virtual Machine Platform** (платформа виртуальных машин) и **Windows Subsystem for Linux** (подсистема Windows для Linux), как показано на рис. 1.3.

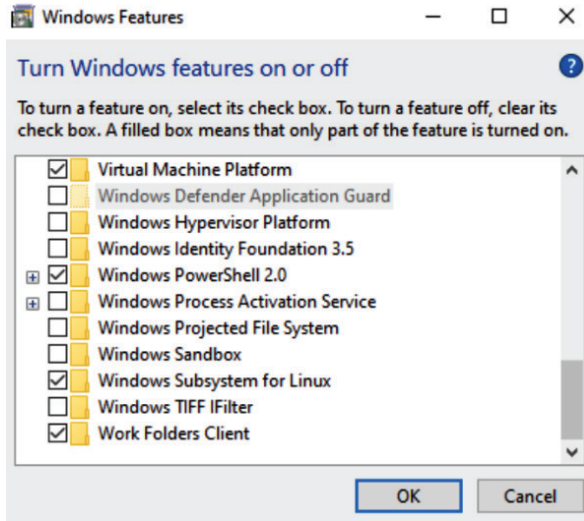


Рис. 1.3 ❖ Включение виртуализации для WSL 2

Для Hyper-V нужно включить **Hyper-V** (рис. 1.4).

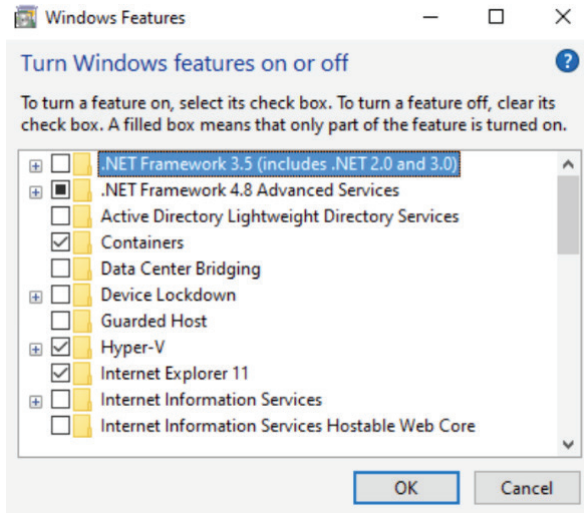


Рис. 1.4 ❖ Включение виртуализации для Hyper-V

Прежде чем продолжить, убедитесь, что ваша учетная запись пользователя добавлена в группу `docker-users`. После этого выйдите из Windows и войдите снова. Запустите Docker и выполните свою первую команду Docker в PowerShell:

```
PS C:\Users\my-user> docker run -d -p 80:80 docker/gettingstarted
Unable to find image 'docker/getting-started:latest' locally
latest: Pulling from docker/getting-started
```

```

59bf1c3509f3: Pull complete
8d6ba530f648: Pull complete
5288d7ad7a7f: Pull complete
39e51c61c033: Pull complete
ee6f71c6f4a8: Pull complete
f2303c6c8865: Pull complete
0645fddcff40: Pull complete
d05ee95f5d2f: Pull complete
Digest: sha256:aa945bdf163395d3293834697fa91fd4c725f47093ec499
f27bc032dc1bdd16
Status: Downloaded newer image for docker/gettingstarted:
latest
852371fcb34fddfe900bddc669af3a7aaab8743f8555fbb9952904bd2516ae7a
PS C:\Users\my-user>

```

Давайте также проверим, установился ли Docker Compose:

```

PS C:\Users\my-user> docker compose version
Docker Compose version v2.2.3

```

Теперь посмотрим, как установить Docker Desktop в Linux.

## ***B Linux***

На момент написания этих строк версия Docker Desktop для Linux была недоступна, но находилась в планах, и это всего лишь вопрос времени, когда она появится. Однако для использования Docker Compose достаточно установить **Docker Engine**.

Наиболее распространенный метод установки – добавить репозитории Docker на рабочую станцию Linux, а затем установить Docker Community Edition с помощью диспетчера пакетов.

Если у вас уже установлена старая версия Docker, то ее следует удалить и установить новые версии `docker-ce` и `docker-ce-cli`. Будем считать, что Docker ранее не был установлен у вас.

Поскольку большой популярностью пользуются дистрибутивы Linux на основе Red Hat, мы установим Docker в Fedora – дистрибутив Linux на основе Red Hat.

Сначала установите пакет `dnf-plugins-core`, содержащий инструменты, которые могут помочь в управлении репозиториями `dnf`:

```
$ sudo dnf -y install dnf-plugins-core
```

Затем добавьте репозиторий `docker-ce` для доступа к двоичным файлам Docker:

```
$ sudo dnf config-manager --add-repo https://download.docker.com/linux/fedora/docker-ce.repo
```

Далее, после настройки репозитория, добавьте пакеты:

```
$ sudo dnf install docker-ce docker-ce-cli containerd.io -y
```

Docker – это демон, который выполняется как служба. Поэтому его запуск производится с помощью команды `systemctl`:

```
$ sudo systemctl start docker
```

Теперь запустите пример `hello-world`:

```
$ sudo docker run hello-world
```

```
Hello from Docker!
```

```
This message shows that your installation appears to be working
correctly.
```

```
...
```

Как видите, почти в каждой команде нам пришлось использовать `sudo`. От этой рутинной обязанности можно избавиться с помощью группы `docker` – пользователи, включенные в нее, получают разрешение на взаимодействие с Docker Engine. Эта группа автоматически создается при установке Docker Engine:

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ docker run hello-world
```

После установки и настройки Docker можно приступить к установке Compose.

Используем для установки ссылку <https://docs.docker.com/compose/install/#install-compose-on-linux-systems>:

```
$ sudo curl -L "https://github.com/docker/compose/releases/
download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /
usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
$ docker-compose --version
docker-compose version 1.29.2, build 5becea4c
```

Здесь можно заметить, что по ссылке доступна более старая версия Compose, чем те, что мы видели в предыдущих разделах. К сожалению, пока что не существует стандартного способа установки Compose V2 в Linux, как, например, в Mac и Windows путем установки Docker Desktop. Однако, поскольку Compose V2 все же можно установить в Linux, мы сделаем это, чтобы далее в книге сосредоточиться исключительно на Compose V2.

Последуем рекомендациям в официальной документации, доступной по адресу <https://docs.docker.com/compose/cli-command/#install-on-linux>:

```
$ mkdir -p ~/.docker/cli-plugins/
$ curl -SL https://github.com/docker/compose/releases/download/
v2.2.3/docker-compose-linux-x86_64 -o ~/.docker/cli-plugins/
docker-compose
$ chmod +x ~/.docker/cli-plugins/docker-compose

$ docker compose version
Docker Compose version v2.2.3
```

## docker compose и docker-compose

В разделах, посвященных установке, можно заметить, что в Linux была установлена `docker compose` для Python.

Аналогичную команду можно найти в Windows, но там она имеет имя `docker-compose`:

```
PS C:\Users\my-user> docker-compose-v1.exe version
docker-compose version 1.29.2, build 5becea4c
docker-py version: 5.0.0
CPython version: 3.9.0
OpenSSL version: OpenSSL 1.1.1g 21 Apr 2020
PS C:\Users\my-user>
```

Первоначально Docker Compose создавался на Python; поэтому в инструкциях по установке упоминается установка пакетов `pip`.

Обратите внимание, что в новых версиях Docker Desktop команда `docker-compose` является псевдонимом для `docker compose`.

Первоначальная версия `docker-compose` по-прежнему поддерживается и развивается. Приложениям Compose, созданным и запущенным с помощью `docker-compose`, доступны вспомогательные инструменты, такие как **Compose Switch** (<https://docs.docker.com/compose/cli-command/#compose-switch>) для упрощения миграции.

При установке Compose Switch старая команда `docker-compose` заменяется командой `compose-switch`.

Compose Switch будет интерпретировать команды, предназначенные для передачи в `docker-compose`, и преобразует их в команды, которые сможет выполнить Compose V2. Затем он вызовет Compose V2 с помощью этой команды.

В нашей книге мы сосредоточимся на версии Compose V2, потому что она является частью `docker-cli`. Эта версия устанавливается по умолчанию в Docker Desktop и поддерживает самые новые функции и дополнительные команды.

Итак, мы установили Docker и Docker Compose и получили некоторое представление о том, как выполнять простые команды. Мы также познакомились с предыдущей версией Compose и узнали, как перейти на последнюю версию. Далее мы погрузимся в исследование особенностей работы Compose и посмотрим, как он взаимодействует с Docker Engine.

## ЗНАКОМСТВО С ПРИНЦИПАМИ РАБОТЫ DOCKER COMPOSE

Установив Docker и Docker Compose, уделим немного времени Compose и узнаем, что это такое и как он работает за кулисами.

На GitHub можно найти проект (<https://github.com/docker/compose>) с исходным кодом Docker Compose. Рассматривая исходный код, можно многое узнать о Compose:

- Compose интегрируется с Docker CLI как плагин;
- Compose взаимодействует с Docker Engine через API;
- Compose предоставляет интерфейс командной строки и преобразует команды в вызовы Docker Engine API;
- Compose читает файл Compose в формате YAML и генерирует описанные в нем ресурсы;
- Compose предоставляет слой для преобразования команд `docker-compose` в CLI-совместимые аналоги;
- Compose взаимодействует с объектами Docker и различает их с помощью меток.

Docker CLI предоставляет API для создания и загрузки плагинов. После создания и загрузки плагина при его вызове ему передается команда CLI:

```
func pluginMain() {
    plugin.Run(func(dockerCli command.Cli) *cobra.Command {
        ...
    })
}

func main() {
    if commands.RunningAsStandalone() {
        os.Args = append([]string{"docker"},
            compatibility.Convert(os.Args[1:])...)
    }
    pluginMain()
}
```

Интерфейс командной строки (CLI) основан на Cobra (<https://github.com/spf13/cobra>) – популярной библиотеке Go для создания приложений командной строки.

Compose, будучи плагином Docker CLI, использует клиента Docker Engine API, предоставляемого Docker CLI:

```
lazyInit.WithService(compose.NewComposeService(dockerCli.Client(), dockerCli.ConfigFile()))
```

Каждая команда, переданная плагину Docker Compose, инициирует взаимодействие с Docker Engine API на нашем хосте. Например, вот как реализована команда `ls`:

```
func (s *composeService)
List(ctx context.Context, opts api.ListOptions) ([]api.Stack, error) {
    list, err := s.apiClient.ContainerList(ctx, moby.ContainerListOptions{
        Filters: filters.NewArgs(hasProjectLabelFilter()),
        All: opts.All,
    })
    if err != nil {
        return nil, err
    }
    return containersToStacks(list)
}
```

Теперь у вас должно сложиться хорошее понимание, как Compose работает и взаимодействует с Docker Engine. Вы также можете обратиться к исходному коду, чтобы получить дополнительную информацию. Далее мы с вами запустим наше первое приложение Docker Compose.

## ВАШ ПЕРВЫЙ ФАЙЛ DOCKER COMPOSE

Представьте, что вам нужно запустить статическую веб-страницу на сервере. Для этой задачи хорошим выбором будет установка сервера NGINX. У нас есть простой HTML-файл `static-site/index.html`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>Hi! This application should run on docker-compose</p>
  </body>
</html>
```

Используя Docker, мы запустим сервер NGINX, взяв официальный образ, доступный по адресу <https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/>:

```
$ docker run --rm -p 8080:80 --name nginx-compose nginx
```

Давайте разберем эту команду:

- Docker Engine запускает Docker-образ сервера NGINX;
- по умолчанию образу назначается порт 80, который отображается в порт 8080 хост-системы, чтобы избежать сложностей с использованием порта из привилегированного диапазона;
- для упрощения взаимодействий с контейнером назначаем ему постоянное имя;
- аргумент `--rm` гарантирует, что после выполнения задачи и остановки контейнер будет удален.

Наш контейнер запущен и работает. Открыв другое окно терминала, мы сможем получить доступ к странице по умолчанию:

```
$ curl 127.0.0.1:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
```

```
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully
installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

У нас получилось запустить сервер NGINX в контейнере. Теперь нам нужно адаптировать команду так, чтобы сервер возвращал нашу HTML-страницу. Для этого достаточно смонтировать файл в контейнер. Завершите выполнение предыдущей команды, нажав комбинацию **Ctrl+C**, а затем введите такую команду:

```
docker run --rm -p 8080:80 --name nginx-compose -v $(pwd)/
static-site:/usr/share/nginx/html nginx
```

Как и ожидалось, сейчас по умолчанию возвращается указанная нами страница:

```
$ curl localhost:8080/index.html
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>Hi! This application should run on docker-compose</p>
  </body>
</html>
$
```

Теперь у нас есть все необходимое для переноса этого приложения в Compose. Для этого создадим файл Compose, который будет устанавливать NGINX:

```
services:
  nginx:
    image: nginx
    ports:
      - 8080:80
```

Рассмотрим его подробнее:

- мы указали имя службы `nginx`;
- для запуска службы будет использоваться тот же образ NGINX;



- для взаимодействий с внешним миром настраиваются те же порты, что и прежде.

Это содержимое следует сохранить в файл с именем `docker-compose.yaml`.

Далее выполним команду Compose в терминале:

```
$ docker compose up
[+] Running 2/0
   Network chapter1_default Created
0.0s
   Container chapter1-nginx-1 Created
0.0s

Attaching to chapter1-nginx-1
chapter1-nginx-1 | /docker-entrypoint.sh: /dockerentrypoint.
d/ is not empty, will attempt to perform
configuration
chapter1-nginx-1 | /docker-entrypoint.sh: Looking for shell
scripts in /docker-entrypoint.d/
...
$
```

HTTP-запрос к серверу возвращает тот же результат, что и при запуске контейнера Docker.

Имя файла играет важную роль. Мы использовали команду Compose, чтобы проанализировать файл Compose, но не указали имя этого файла. Так же, как в случае с командой `docker build` и файлом `Dockerfile`, команда `docker compose` будет искать файл с именем `docker-compose.yaml` в текущем каталоге. Если файл существует, он будет использован как файл Compose по умолчанию. Но имейте в виду, что мы не ограничены единственным именем файла; мы можем использовать любое другое имя файла для наших приложений Compose. В следующих главах вы увидите случаи использования других имен для файлов Compose и будете запускать приложение, используя параметр `-f`.

Далее мы смонтируем свою HTML-страницу в конфигурации Compose:

```
services:
  nginx:
    image: nginx
    ports:
      - 8080:80
    volumes:
      - ./static-site:/usr/share/nginx/html
```

Какой бы простой ни казалась предыдущая команда Docker, за кулисами она создает том Docker, согласно указанному пути в файловой системе, а затем подключает его к контейнеру. То же относится и к Compose. Здесь мы определяем том, указав путь в нашей файловой системе, который затем монтируется в каталог контейнера:

```
$ curl localhost:8080/index.html
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Hello World</title>
</head>
<body>
  <p>Hi! This application should run on docker-compose</p>
</body>
</html>
```

Как и ожидалось, результат получился тем же, что и в примере с Docker.

Итак, в этом разделе мы запустили экземпляр NGINX с помощью утилиты командной строки Docker, а затем проделали то же самое с помощью Compose, добавив соответствующие разделы YAML, определяющие параметры для команды Docker. Теперь мы можем сделать следующий шаг – создать и запустить образ Docker в Docker Compose.

## ИСПОЛЬЗОВАНИЕ ОБРАЗА DOCKER В DOCKER COMPOSE

Используя Compose, мы запустили образ NGINX и изменили HTML-страницу, отображаемую по умолчанию. Начав применять Compose, мы продолжим работать с этим инструментом и попробуем создать свой образ Docker.

Далее мы создадим образ NGINX, который выводит журнальные записи в формате JSON. Этот формат поддерживается такими инструментами, как **CloudWatch** (<https://aws.amazon.com/cloudwatch/>), **StackDriver** (<https://cloud.google.com/products/operations>) и **ELK Stack** (<https://www.elastic.co/elastic-stack/>), которые предлагают дополнительные возможности просмотра журналов с поиском и фильтрацией записей по полям в элементах JSON.

Для решения этой задачи мы должны определить, как в NGINX задается текущий формат журналирования. Поскольку у нас уже есть готовый контейнер, мы запустим его с помощью Compose и проверим конфигурацию:

```
$ docker ps
CONTAINER ID   IMAGE     COMMAND                    CREATED
STATUS        PORTS    NAMES
dc0ca7ebe0cb  nginx    "/docker-entrypoint...."  7 hours ago
Up 7 hours    0.0.0.0:8080->80/tcp    chapter1-nginx-1
$ docker exec -it chapter1-nginx-1 cat /etc/nginx/nginx.conf
```

```
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
```

```

    worker_connections 1024;
}
http {
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                   '$status $body_bytes_sent "$http_referer" '
                   '"$http_user_agent" "$http_x_forwarded_for"';

```

Найдя работающий контейнер с помощью `docker ps` и выполнив команду `cat` в командной оболочке контейнера, мы получили текущую настройку `log_format`, как она определена в конфигурационном файле `/etc/nginx/nginx.conf` в контейнере. Мы изменим этот формат на JSON и создадим свой образ Docker с этой настройкой.

Для этого скопируем файл в локальную систему, чтобы применить изменения:

```
$ docker cp chapter1-nginx-1:/etc/nginx/nginx.conf nginx.conf
```

Отредактируем `nginx.conf`, задав в параметре `log_format` значение `json`:

```

log_format main escape=json '{"remote_addr":'$remote_
addr',"remote_user":'$remote_user',"time":'[$time_
local]',"request":'$request',
                        '"status":'$status',"body_bytes_
sent":'$body_bytes_sent',"http_referer":'$http_referer',
                        '"http_user_agent":'$http_user_
agent',"http_x_forwarded_for":'$http_x_forwarded_for'}';

```

Теперь содержимое файла выглядит так:

```

user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main escape=json '{"remote_addr":'$remote_
addr',"remote_user":'$remote_user',"time":'[$time_
local]',"request":'$request',
                        '"status":'$status',"body_bytes_
sent":'$body_bytes_sent',"http_referer":'$http_referer',

```

```

        "http_user_agent": "${http_user_
agent", "http_x_forwarded_for": "${http_x_forwarded_for}"}';

    access_log /var/log/nginx/access.log main;
    sendfile      on;
    #tcp_nopush   on;

    keepalive_timeout 65;

    #gzip on;

    include /etc/nginx/conf.d/*.conf;
}

```

Получив необходимый конфигурационный файл, создадим базовый образ NGINX, который будет использовать эту конфигурацию. Для этого создадим следующий файл Dockerfile:

```

FROM nginx

COPY nginx.conf /etc/nginx/nginx.conf

```

и сгенерируем сам образ:

```
$ docker build -t custom-nginx:0.1 .
```

Давайте продолжим и используем этот образ с недавно созданным файлом `docker-compose.yaml`:

```

services:
  nginx:
    image: custom-nginx:0.1
    ports:
      - 8080:80
    volumes:
      - ./static-site:/usr/share/nginx/html

```

```
$ docker compose up
```

```

...
chapter1-nginx-1 | 2022/02/10 08:09:27 [notice] 1#1: start
worker process 33
chapter1-nginx-1 | {"remote_addr":"172.19.0.1","remote_
user":"","time":"[10/Feb/2022:08:09:33 +0000]","request":"GET
/ HTTP/1.1","status":"200","body_bytes_sent":"177","http_
referer":"","http_user_agent":"curl/7.77.0","http_x_forwarded_
for":""}
...

```

На данный момент Compose успешно работает с нашим приложением в нашем собственном образе Docker. До сих пор инструмента Compose было достаточно для использования нашего собственного образа с дополнительными изменениями в конфигурации. Результаты получились в точности такими же, как при запуске приложения с помощью команд Docker.

## Итоги

В этой главе мы познакомились с Docker Compose и некоторыми его наиболее примечательными функциями. Мы установили Compose в разные операционные системы и определили различия между установками. Затем поговорили о различных версиях Compose, Docker-Compose V1 и Docker Compose V2, а также выбрали версию, которая будет использоваться в этой книге. Заглянув в исходный код Compose, мы сделали еще один шаг и проверили, как Compose работает и взаимодействует с интерфейсом командной строки Docker. Затем мы запустили приложение Docker с помощью команды `docker` и создали его эквивалент в Compose. Следующим шагом мы настроили образ, который использовали в первом примере, и развернули с помощью Compose.

Во второй главе мы с помощью Compose создадим приложение, которое будет работать и взаимодействовать с базой данных Redis.