СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Благодарности

В первую очередь я хочу поблагодарить мою жену Лену за терпение и поддержку, проявленные во время написания этой книги, как и всех предыдущих.

Также традиционная большая благодарность рецензентам книги.

Об авторе

Андрей Маркелов имеет пятнадцатилетний опыт преподавания как авторских курсов, так и авторизированных курсов по ИТ-технологиям таких компаний, как Red Hat и Microsoft.

Последние годы автор работает в качестве старшего системного архитектора компании Ericsson, специализируясь на облачных технологиях, инфраструктуре виртуализации сетевых функций (NFV-I) и технологиях управления контейнерами. До этого работал в качестве старшего системного архитектора в компании Red Hat, а также в крупных системных интеграторах России, получив более чем десятилетний опыт продаж, проектирования и внедрения сетевых и инфраструктурных решений.

Около шестидесяти публикаций в российских ИТ-журналах («Системный администратор», «Linux Format», «PC Week» и др.). Автор книг «OpenStack. Практическое введение в облачную операционную систему» (ДМК Пресс, 2015, 2016, 2017, 2018 гг.), выдержавшей четыре издания, и вышедшей в 2016 году книги издательства Apress «Certified OpenStack Administrator Study Guide».

Андрей является сертифицированным архитектором Red Hat (RHCA) с 2009 года и имеет сертификаты Red Hat в таких технологиях, как OpenStack, OpenShift, RHV, CloudForms, Ansible, GlusterFS, настройка производительности, безопасности Linux-систем и др. Кроме того, автор имеет сертификаты Microsoft Certified System Engineer, Sun Certified System Administrator, Novell Certified Linux

Professional, Linux Professional Institute Certification (LPIC-1), Mirantis Certified OpenStack Administrator, OpenStack Foundation Certified OpenStack Administrator, Cisco Certified Network Associate.

Блог автора располагается по адресу: http://markelov.blogspot.com/. Его twitter-аккаунт @amarkelov.

Рецензенты

Артемий Кропачев (Artemii Kropachev) – технический директор (CTO) и главный ИТ-архитектор (Principal Architect) компании «Li9 Technology Solutions», расположенной в г. Финикс, США. Артемий имеет более 15 лет опыта в ИТ, который покрывает проектирование комплексных ИТ-систем, разработку программного обеспечения, SDN/NFV, облачные решения и системы автоматизации. Артемий занимается проектированием и внедрением проектов на базе технологий Red Hat. Основной фокус направлен на контейнеризацию приложений на базе Red Hat OpenShift Container Platform. Артемий является автором книг «Ansible for IT Experts», «Learn OpenShift», «Learn DevOps», «Learn Linux». Кроме того, Артемий на 2019 г. является обладателем самого высокого сертификационного статуса по инфраструктурным продуктам Red Hat в мире – Red Hat Certified Architect Level XX in Infrastructure.

Василий Ангапов (Vasiliy Angapov) – старший архитектор компании Bell Integrator, г. Москва. Василий имеет более 10 лет опыта в ИТ, долгое время занимался внедрением облачных решений компании Red Hat. В настоящее время успешно занимается проектированием и внедрением различных микросервисных приложений на базе Kubernetes в облачных платформах Google Cloud и Amazon Web Services.

Предполагаемая аудитория

Данная книга рассчитана на ИТ-специалистов, работающих с GNU/Linux и желающих познакомиться с технологиями контейнеров и системой оркестрации Kubernetes.

От читателя требуются базовые навыки работы с операционной системой GNU/Linux. Умение работать в командной строке и знание основных команд GNU/Linux обязательны.

О чем эта книга

Книга состоит из восьми глав и знакомит читателя с механизмами, обеспечивающими работу контейнеров в GNU/Linux, основами работы с контейнерами при помощи Docker и Podman, а также системой оркестрирования контейнеров Kubernetes. Помимо этого, книга знакомит с особенностями одного из самых популярных дистрибутивов Kubernetes – OpenShift (OKD).

Глава 1. Введение в контейнеры GNU/Linux

Первая глава вводит читателя в предметную область и историю технологий контейнеров в GNU/Linux. Также обсуждается архитектура «движка» Docker. В этой главе читатель создает лабораторный стенд и устанавливает Docker в GNU/Linux на примере дистрибутива CentOS 7.

Глава 2. Основы работы с контейнерами Docker

Во второй главе читатель познакомится с базовыми операциями и основными командами утилиты docker. Рассмотрены такие темы, как поиск образов контейнеров и теги, запуск контейнеров, управление состоянием контейнеров и их изоляция, работа с хранилищем, сетью и реестром.

Глава 3. Создание контейнеров при помощи Dockerfile

В этой главе обсуждается файл инструкций для сборки контейнеров Dockerfile и сам процесс создания новых образов.

Глава 4. Работа с контейнерами Docker без движка Docker

В данной главе рассматриваются инструменты, изначально разработанные для альтернативной среды исполнения контейнеров CRI-O: podman, buildah и skopeo. Все три утилиты не требуют наличия запущенного демона и независимы от Docker.

Глава 5. Введение в Kubernetes и установка кластера

В пятой главе читатель познакомится с архитектурой системы оркестрации контейнеров Kubernetes и создаст кластер, состоящий из трех узлов, для последующих экспериментов в процессе обучения.

Глава 6. Основы работы с Kubernetes

В данной главе описаны основные объекты Kubernetes. Читатель на практике познакомится с созданием pod-модулей, внедрений и их масштабированием. Рассмотрена работа с томами

и запросами на постоянные тома, а также секреты, карты конфигурации и сервисы Kubernetes.

Глава 7. Расширенные возможности Kubernetes

Седьмая глава знакомит с контроллерами StatefulSet, DaemonSet, Job и CronJob. Также рассмотрена работа с Ingress и Ingress Controller.

Глава 8. Знакомство с OpenShift и OKD

В данной главе читатель знакомится с одним из популярных дистрибутивов Kubernetes, разрабатываемых компанией Red Hat, – OpenShift/OKD. Дается сравнение OpenShift и Kubernetes, показана установка тестовой среды при помощи команды oc cluster up, и продемонстрирована работа с основными объектами OpenShift.

Глава 1.
ВВЕДЕНИЕ В КОНТЕЙНЕРЫ GNU/LINUX

В первой главе мы поговорим об истории технологий контейнеров в GNU/Linux и архитектуре «движка» Docker.

Отличия контейнеров от виртуализации

В отличие от «вертикального» абстрагирования в случае виртуализации, контейнеры, в частности контейнеры Docker, с рассмотрения которых мы начнем, обеспечивают «горизонтальное» разбиение операционной системы на отдельные изолированные окружения. За счет того, что в каждом контейнере, в отличие от виртуализации, обходятся без использования отдельного экземпляра операционной системы, значительно ниже накладные расходы. Минусом является тот факт, что вы не сможете на одном узле в контейнерах запускать разные операционные системы, например Windows и GNU/Linux.

Также необходимо отметить, что зачастую контейнеры используются поверх виртуальных машин, которые могут располагаться как в частном, так и в публичном облаке.

Итак, контейнеры обеспечивают часть преимуществ виртуальных машин, при этом используя меньше аппаратных ресурсов. Преимущества использования контейнеров, по сравнению с виртуальными машинами, следующие:

- горизонтальная изоляция приложений;
- меньшие накладные расходы на инфраструктуру;
- изоляция не на уровне операционной системы, а на уровне окружения приложения (необходимые библиотеки);
- скорость развертывания приложений и скорость перезапуска приложений;
- параллельная работа нескольких окружений одного приложения;

- «виртуализация приложений» – независимость приложения от операционной системы и библиотек, что упрощает обновления и тестирование;
- возможность переиспользования одного и того же контейнера для нескольких приложений на одном узле.

При использовании контейнеров, и контейнеров Docker в частности, нужно иметь в виду следующие соображения:

- Docker может являться основой для облака, работающего по сервисной модели PaaS, но сам по себе не является таким облаком. Примеры облачных платформ модели PaaS: Cloud Foundry, Red Hat OpenShift (OKD);
- хотя с технической точки зрения Docker всего лишь меняет способ упаковки приложений, это может повлечь за собой изменение архитектуры приложения;
- контейнеры – это не противопоставление виртуализации. Контейнеры могут и довольно часто запускаются внутри виртуальных машин. В качестве примера можно привести PaaS OpenShift Online, в котором контейнеры работают внутри инфраструктуры IaaS Amazon.

Все контейнеры на одном узле используют одно и то же ядро операционной системы. С точки зрения ядра операционной системы GNU/Linux, не существует такого понятия, как «контейнер». Контейнеры – это просто процессы GNU/Linux, использующие стандартные механизмы изоляции ядра.

Набор утилит, обеспечивающих работу с контейнерами, называется «движком» (container engine), а его часть, отвечающая за запуск и мониторинг контейнеров, – средой выполнения (container runtime). Мы начнем рассмотрение контейнеров с движка Docker, однако в дальнейшем, когда перейдем к Kubernetes, также познакомимся с альтернативой – CRI-O. Среда выполнения присваивает контейнерам собственные идентификаторы (ID), но этот уровень абстракции ничего не значит для ядра операционной системы. Движок контейнеров также предоставляет такие полезные вещи, как возможность диагностики и запуска дополнительных процессов в том же самом окружении контейнера, ссылаясь на его идентификатор.

История Docker

Готовые к промышленному применению контейнеры на открытых технологиях в GNU/Linux появились позже, чем механизм Jails во FreeBSD (2000 год) или Zones в Solaris (2005 год). Долгое время самой известной в GNU/Linux контейнерной технологией оставался коммерческий продукт Virtuozzo (с 2001 года) компании, в разные годы называвшейся Swsoft, Parallels и Odin. Часть кода Virtuozzo была открыта в 2005 году в виде проекта OpenVZ. Примерно с того же времени развивается проект Linux-VServer, требующий патчей к ядру Linux (нулевая версия появилась в 2001 году).

В 2006 году появился механизм cgroups (от англ. control groups – ресурсные, или контрольные, группы) как редизайн существующей технологии cpuset (подробнее о cpuset в блоге автора книги в заметке 2009 года [1]). В 2008 году появились пользовательские пространства имен, которые стали еще одной составной частью контейнеров GNU/Linux. Про контрольные группы и пространства имен мы поговорим далее в книге.

В 2008 году инженерами IBM был инициирован проект LXC, который долгое время в первую очередь ассоциировался с открытыми контейнерами GNU/Linux в противовес проприетарному Virtuozzo. Первая версия LXC вышла только в 2014 году, получив в том числе поддержку системы мандатного контроля доступа (MAC) SELinux.

В июле 2015 года Linux Foundation анонсировала запуск нового проекта Open Container Project (OCP) [21], который призван установить общие стандарты и обеспечить фундамент совместимости для продолжения развития контейнерных решений без дальнейшей фрагментации этого направления. Инициативу OCP поддержали многие крупные компании и организации, среди которых можно упомянуть Amazon Web Services, Apcera (в дальнейшем компания поглощена компанией Ericsson), Cisco, CoreOS (поглощена компанией Red Hat), EMC, Google, HP, IBM, Intel, Microsoft, Red Hat (теперь принадлежит IBM), VMware и др. В качестве основы Open Container Project выступили в значительной степени наработки проекта и компании Docker.

На настоящий момент OCI поддерживает две спецификации:

- спецификация среды выполнения (runtime-spec);
- спецификация образов контейнеров (image-spec).

В июле 2015 года была выпущена первая версия системы оркестрации контейнеров Kubrnetes, и краткий экскурс в историю мы продолжим в соответствующей главе.

Архитектура Docker

Docker использует клиент-серверную архитектуру и состоит из клиента – утилиты docker, которая обращается к серверу при помощи REST API, и демона в операционной системе GNU/Linux (dockerd). Хотя Docker работает и в отличных от GNU/Linux операционных системах, в этой книге они не рассматриваются.

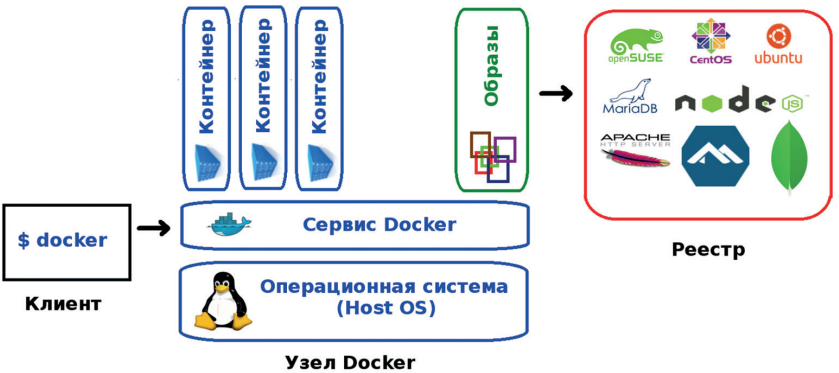

Рис. 1.1 ❖ Архитектура контейнеров Docker в GNU/Linux

Основные компоненты Docker:

- **контейнеры** – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения. Проще всего дать определение контейнеру Docker как запущенному из образа приложению. Кстати, именно этим идеологически и отличается Docker, например, от LXC (Linux Containers), хотя они используют одни и те же технологии ядра Linux. Разработчики проекта Docker исповедуют принцип: один контейнер – это одно приложение;
- **образы** – доступные только для чтения шаблоны приложений. Поверх существующих образов могут добавляться новые уровни образов, которые совместно представляют

файловую систему, изменяя или дополняя предыдущий уровень. Обычно новый образ создается либо при помощи сохранения уже запущенного контейнера в новый образ поверх существующего, либо при помощи специальных инструкций для утилиты Dockerfile. Для разделения различных уровней контейнера на уровне файловой системы могут использоваться UnionFS, aufs, btrfs, vfs, OverlayFS и Device Mapper;

- **реестры** (registry), содержащие репозитории (repository) образов, – сетевые хранилища образов. Могут быть как приватными, так и общедоступными. Самым известным реестром является Docker Hub [3]. В книге также упоминаются репозитории, содержащие rpm-пакеты. С ними работают команды yum и dnf. Не путайте репозитории, служащие для установки пакетов операционной системы, и репозитории Docker.

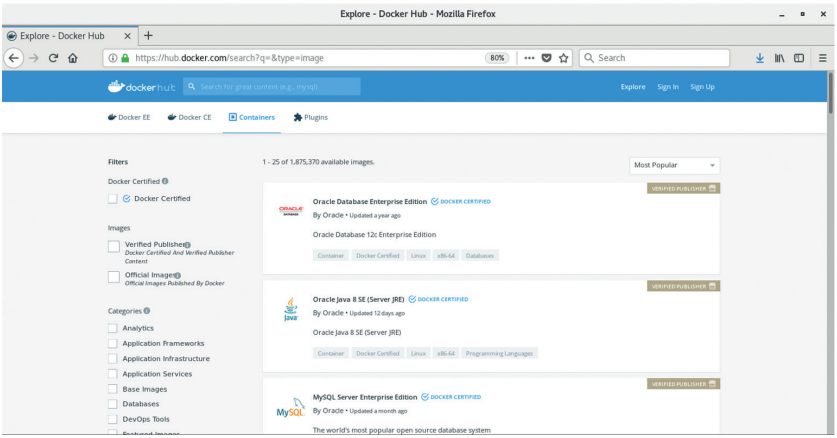

Рис. 1.2 ❖ Реестр образов Docker Hub

Для изоляции контейнеров и обеспечения безопасности в операционных системах GNU/Linux используются стандартные технологии ядра Linux, такие как:

- пространства имен (Linux Namespaces);
- контрольные группы (cgroups);
- средства управления привилегиями (Linux Capabilities);
- дополнительные, мандатные системы обеспечения безопасности, такие как AppArmor или SELinux.

Рассмотрим перечисленные технологии чуть более подробно.

Механизм контрольных групп (cgroups) предоставляет инструмент для тонкого контроля над распределением, приоритизацией и управлением системными ресурсами. Контрольные группы реализованы в ядре Linux. В современных дистрибутивах управление контрольными группами реализовано через systemd, однако сохраняется возможность управления при помощи библиотеки libcgroup и утилиты cgconfig. Основные иерархии контрольных групп (их также называют контроллерами) перечислены ниже:

- **blkio** – задает лимиты на операции ввода-вывода и на доступ к блочным устройствам;
- **cpu** – используя планировщик процессов, распределяет процессорное время между задачами;
- **cpuacct** – создает автоматические отчеты по использованию ресурсов центрального процессора. Работает совместно с контроллером cpu, описанным выше;
- **cpuset** – закрепляет за задачами определенные процессоры и узлы памяти;
- **devices** – регулирует доступ задач к определенным устройствам;
- **freezer** – приостанавливает или возобновляет задачи;
- **memory** – устанавливает лимиты и генерирует отчеты об использовании памяти задачами контрольной группы;
- **net_cls** – осуществляет тегирование сетевых пакетов идентификатором класса (classid). Это позволяет контроллеру трафика (команда tc) и брандмауэру (iptables) учитывать данные теги при обработке трафика;
- **perf_event** – позволяет производить мониторинг контрольных групп при помощи утилиты perf;
- **hugetlb** – дает возможность использовать виртуальные страницы памяти большого размера и применять к ним лимиты.

Пространства имен, в свою очередь, контролируют не распределение ресурсов, а доступ к структурам данных ядра. Фактически это означает изоляцию процессов друг от друга и возможность иметь параллельно «одинаковые», но не пересекающиеся друг с другом иерархии процессов, пользователей и сетевых интерфейсов. При желании разные сервисы могут иметь даже свои собственные loopback-интерфейсы.

Примеры пространств имен, используемых Docker:

- **PID, Process ID** – изоляция иерархии процессов;
- **User** – изоляция идентификаторов пользователей (UID) и групп (GID);
- **NET, Networking** – изоляция сетевых интерфейсов;
- **IPC, InterProcess Communication** – управление взаимодействием между процессами;
- **MNT, Mount** – управление точками монтирования;
- **UTS, Unix Timesharing System** – изоляция ядра, идентификаторов версии, имени хоста и доменного имени NIS.

Механизм под названием Capabilities позволяет разбить привилегии пользователя root на небольшие группы привилегий и назначать их по отдельности. Данный функционал в GNU/Linux появился начиная с версии ядра 2.2. Изначально контейнеры запускаются уже с ограниченным набором привилегий. При помощи опций команды docker вы можете разрешать и запрещать такие действия, как:

- операции монтирования;
- доступ к сокетам;
- выполнение части операций с файловой системой, например изменение атрибутов файлов или владельца.

Подробнее ознакомиться с привилегиями можно при помощи man-страницы руководства GNU/Linux CAPABILITIES(7).

Перейдем к последнему опциональному компоненту контейнерных технологий. SELinux (Security-Enhanced Linux) – это реализация системы мандатного контроля доступа (MAC), которая может работать (и работает по умолчанию) параллельно с классической дискреционной системой контроля доступа (DAC).

Оставаясь в рамках DAC, мы имеем фундаментальное ограничение в плане разделения доступа пользователей к ресурсам – доступ к ресурсам основывается на правах доступа пользователя. Это классические права RWX на трех уровнях – владелец, группа-владелец и остальные. Плюс к этому POSIX ACL, которые лишь расширяют число уровней, на которых можно определить права, но не более.

Таким образом, любое приложение, запущенное с правами user1, теоретически может сделать все, что угодно, со всеми данными, к которым имеет доступ user1. И не важно, например, что

данное приложение – это почтовая программа, которой нужно иметь доступ только к письмам user1. Эта программа будет иметь доступ и, например, к видеофайлам user1, и к картинкам user1. И мало того, что сама программа имеет доступ, но она же может и все эти данные сделать доступными остальным. Все становится гораздо хуже, когда user1 имеет UID, равный 0 (то есть это root). Мы утыкаемся в классическую «проблему суперпользователя». В данном случае мы получаем всего лишь два уровня доступа: root и обыкновенные пользователи. Иными словами, становится невозможным реализовать доступ с использованием минимально необходимых привилегий.

В MAC же права доступа определяются самой системой при помощи специально определенных политик. Если же говорить конкретно о рассматриваемой реализации – SELinux, то такие политики работают на уровне системных вызовов и применяются самим ядром.

SELinux действует после классической модели безопасности Unix. Иными словами, через SELinux нельзя разрешить то, что запрещено через права доступа пользователей/групп. Политики описываются при помощи специального гибкого языка описания правил доступа. В большинстве случаев правила SELinux «прозрачны» для приложений и не требуется никакой их модификации.

Установка Docker в GNU/Linux на примере CentOS 7

В данной книге для примеров мы будем использовать дистрибутив CentOS 7. Инструкции также должны без изменений работать на любых других производных от Red Hat Enterprise Linux 7. Единственное возможное отличие – это порядок настройки доступа к репозиториям RPM-пакетов.

Первое, что необходимо, – установить саму операционную систему. Предполагается, что это не должно вызвать затруднений у читателя. В качестве варианта установки можно выбрать Minimal или Server with GUI.

После установки операционной системы обновите все установленные пакеты командой

```
# yum -y update
```

При работе с CentOS у вас есть выбор: использовать последнюю версию из upstream или версию, собранную проектом CentOS, с дополнениями Red Hat. Описание изменений доступно на странице по ссылке [4]. В основном это обратное портирование исправлений из новых версий upstream и изменения, предложенные разработчиками Red Hat, но пока не принятые в основной код. Для целей первоначального знакомства с Docker мы будем использовать версию по умолчанию из стандартного репозитория CentOS:

```
[root@centos7 ~]# yum -y install docker
...
[root@centos7 ~]# yum info docker
Installed Packages
Name        : docker
Arch        : x86_64
Epoch       : 2
Version     : 1.13.1
Release     : 88.git07f3374.el7.centos
Size        : 65 M
Repo        : installed
From repo   : extras
Summary     : Automates deployment of containerized applications
URL         : https://github.com/docker/docker
License     : ASL 2.0
...
```

Настройки репозитория для установки upstream-версии, как и инструкции для инсталляции в других дистрибутивах и операционных системах, приведены в руководстве по инсталляции по ссылке [5].

Запускаем и включаем сервис:

```
[root@centos7 ~]# systemctl start docker.service
[root@centos7 ~]# systemctl enable docker.service
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.
service to /usr/lib/systemd/system/docker.service.
```

Проверяем статус сервиса:

```
[root@centos7 ~]# systemctl status docker.service
• docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor
preset: disabled)
   Active: active (running) since Sun 2018-12-3013:04:04 EST; 5min ago
     Docs: http://docs.docker.com
 Main PID: 12844 (dockerd-current)
```

```
   CGroup: /system.slice/docker.service
           ├─12844 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/
libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt
native.cgroupdriver=systemd --userl...
           └─12848 /usr/bin/docker-containerd-current -l unix:///var/run/
docker/libcontainerd/docker-containerd.sock --metrics-interval=0 --start-
timeout 2m --state-dir /var/run/docker/l...
```

Также можно посмотреть системную информацию о Docker и окружении:

```
[root@centos7 ~]# docker info
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: xfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
Init Binary: /usr/libexec/docker/docker-init-current
containerd version:  (expected: aa8187dbd3b7ad67d8e5e3a15115d3eef43a7ed1)
runc version: N/A (expected: 9df8b306d01f59d3a8029be411de015b7304dd8f)
init version: fec3683b971d9c3ef73f284f176672c44b448662 (expected:
949e6facb77383876aeff8a6944dde66b3089574)
Security Options:
 seccomp
  WARNING: You're not using the default seccomp profile
  Profile: /etc/docker/seccomp.json
 selinux
Kernel Version: 3.10.0-957.1.3.el7.x86_64
Operating System: CentOS Linux 7 (Core)
OSType: linux
Architecture: x86_64
Number of Docker Hooks: 3
CPUs: 1
Total Memory: 3.701 GiB
Name: centos7.test.local
```

```
ID: OHUV:5U3E:EOFY:LROQ:MCB5:55Q2:QSVK:DHLN:4AAZ:V22X:F7FV:B7NA
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
Registries: docker.io (secure)
```

Опции запуска демона, как это обычно бывает в CentOS, хранятся в файлах директории /etc/sysconfig/. В данном случае это файлы docker*:

```
[root@centos7 ~]# grep  'OPTIONS' /etc/sysconfig/docker*
/etc/sysconfig/docker:OPTIONS='--selinux-enabled --log-driver=journald
--signature-verification=false'
/etc/sysconfig/docker-network:DOCKER_NETWORK_OPTIONS=
/etc/sysconfig/docker-storage:DOCKER_STORAGE_OPTIONS="--storage-driver
overlay2 "
```

Как мы видим, Docker запускается с поддержкой SELinux, драйвером журналирования journald и драйвером хранилища overlay2.

Помимо показанной выше команды docker info, для того чтобы ознакомиться с версиями вашего окружения Docker, будет полезным и вывод docker version:

```
[root@centos7 ~]# docker version
Client:
 Version:         1.13.1
 API version:     1.26
 Package version: docker-1.13.1-88.git07f3374.el7.centos.x86_64
 Go version:      go1.9.4
 Git commit:      07f3374/1.13.1
 Built:           Fri Dec  716:13:512018
 OS/Arch:         linux/amd64

Server:
 Version:         1.13.1
 API version:     1.26 (minimum version 1.12)
 Package version: docker-1.13.1-88.git07f3374.el7.centos.x86_64
 Go version:      go1.9.4
 Git commit:      07f3374/1.13.1
 Built:           Fri Dec  716:13:512018
 OS/Arch:         linux/amd64
 Experimental:    false
```

В следующей главе мы научимся запускать контейнеры Docker и изучим базовые возможности движка контейнеров.

Вопросы для самопроверки

1. **Назовите отличия использования контейнеров по сравнению с виртуализацией (укажите все правильные ответы).**

A. Меньшие накладные расходы на инфраструктуру
B. Время старта приложений больше
C. Невозможность запуска GNU/Linux- и Windows-приложений на одном хосте
D. Обязательное использование гипервизора KVM

2. **Назовите основные компоненты Docker (укажите все правильные ответы).**

A. Гипервизор
B. Контейнеры
C. Образы виртуальных машин
D. Реестры

3. **Какие технологии используются для работы с контейнерами в GNU/Linux (укажите все правильные ответы)?**

A. Пространства имен (Linux Namespaces)
B. Подключаемые модули аутентификации (PAM)
C. Контрольные группы (cgroups)
D. Аппаратная поддержка виртуализации

Список ссылок

1. http://markelov.blogspot.se/2009/01/blog-post.html
2. https://www.opencontainers.org/
3. https://hub.docker.com/
4. http://www.projectatomic.io/docs/docker_patches/
5. https://docs.docker.com/install/linux/docker-ce/centos/