УДК 004.773:614.29 (045)
Мельник В.М., Пех П.А., Мельник М.М.
Луцький національний технічний університет

LINUX-КОНТЕЙНЕРИ І МАЙБУТНІЙ CLOUD

Melnyk V., Pekh P., Melnyk M. Linux containers and the future cloud. Linux-based containers are described in this article, and briefly, explanations of the underlying c-groups and namespaces kernel features are put in the article. Some Linux-based container projects are also discussing here, focusing on the promising and popular LXC (LinuX Containers) project. There also is a look at the LXC-based Docker engine, which provides an easy and convenient way to create and deploy LXC containers. Several hands-on examples showed how simple it is to configure, manage and deploy LXC containers with the userspace LXC tools and the Docker tools.

Due to the advantages of the LXC and the Docker open-source projects, and due to the convenient and simple tools to create, deploy and configure LXC containers, as described in this article, we presumably will see more and more cloud infrastructures that will integrate LXC containers instead of using virtual machines in the near future. However, as explained in this article, solutions like Xen or KVM [1] have several advantages over Linux-based containers and still are in need, so they probably will not disappear from the cloud infrastructure in the next few years.

Keywords: Linux-based container, kernel features, Docker engine, userspace LXC tools, Docker tools, cloud infrastructures

Мельник В.М., Пех П.А., Мельник М.М. Linux-контейнери і майбутній cloud. В роботі описані контейнери на базі операційної системи Linux, а також поданий детальний опис ведучих С-груп та можливості просторів імен ядра. Деякі контейнерні проекти на базі Linux також обговорюються з погляду їх перспектив застосування та актуальності в LXC проектах. Також обговорюється погляд на Docker engine на базі LXC, який забезпечує легкий та зручний шлях для створення і розширення LXC-контейнерів. Багато створених зручних варіантів розробок довели, як просто конфігурувати, управляти і розширювати LXC-контейнери засобами LXC простору користувача і засобами Docker.

В зв'язку з перевагами LXC- і Docker-проектів з доступними ресурсами і в зв'язку з зручністю та простотою засобів створення, поширення та конфігурації LXC-контейнерів, як описано в цій роботі, з наполегливістю більше і більше розглядатимуться cloud-інфраструктури, які в близькому майбутньому здійснюватимуть інтеграцію LXC-контейнерів замість використання віртуальних машин. Однак, як пояснюється в даній статті, висновки подібні Xen чи KVM [1] мають деякі переваги над базовими Linux-контейнерами і, все ж, є потрібними, а тому не можуть просто зникнути з cloud-інфраструктури протягом декількох наступних років.

Ключові слова: контейнери на базі операційної системи Linux, можливості ядра, Docker engine, засоби LXC простору користувача, засобами Docker, cloud-інфраструктури

Мельник В.М., Пех П.А., Мельник М.М. Linux-контейнеры і будущий cloud. В работе описаны контейнеры на базе операционной системы Linux, а также подано детальное описание основных С-групп и возможности пространств имен ядра. Некоторые контейнерные проекты на базе Linux также обсуждаются с точки зрения их перспектив использования и актуальности в LXC проектах. Также обсуждается взгляд на Docker engine на базе LXC, который обеспечивает легкий та выгодный путь для создания и расширения LXC-контейнеров. Много созданных удобных вариантов разработок довели, как просто конфигурировать, управлять и расширять LXC-контейнеры посредствами LXC пространства пользователя і средствами Docker.

В связи с преимуществами LXC- і Docker-проектов с открытыми ресурсами и в связи с удобностью и простотой средств создания, расширения та конфигурации LXC-контейнеров, как описано в этой роботе, з намерением больше и больше рассматриваются cloud-инфраструктуры, которые в близком будущем будут производить интеграцию LXC-контейнеров вместо использования виртуальных машин. Все же, как объясняется в данной статье, выводы подобны Xen или KVM [1] имеют некоторые преимущества над базовыми Linux-контейнерами и, все ж, есть нужными, а поэтому не могут просто исчезнуть з cloud-инфраструктуры на протяжении нескольких следующих лет.

Ключевые слова: контейнеры на базе операционной системы Linux, возможности ядра, Docker engine, средства LXC пространства пользователя, средства Docker, cloud-инфраструктуры

Постановка проблеми. Контейнерна інфраструктура на базі Linux являє собою cloud-технологію, засновану на швидкій і зручній віртуалізації процесів [1]. Вона забезпечує своїм користувачам середовище найбільш близьке до стандартного Linux дистрибутива. В протилежність рішенням пара-віртуалізації (Xen) та рішенням апаратної віртуалізації (KVM), які підтримуються віртуальними машинами (ВМ), контейнери не створюють екземплярів ядра операційних систем. У зв'язку з тим фактом, що контейнери є легше застосовні, ніж ВМ, то можна домогтися більш високого зближення Linux з контейнерами, ніж з віртуальними машинами на одному і тому ж хості. З практичної точки зору, можна розгорнути більше примірників контейнерів, ніж віртуальних машин на одному і тому ж хості.

© Мельник В.М., Пех П.А., Мельник М.М.

Іншою перевагою контейнерів над ВМ є те, що запуск і зупинка контейнера здійснюється набагато швидше, ніж запуск і вимикання ВМ. Всі контейнери для хоста виконуються під тим же ядром, на відміну від рішень для віртуалізації, таких як Xen або KVM, де кожна віртуальна машина володіє власним ядром. Іноді обмеження виконання під тим же ядром у всіх контейнерах одного і того ж хоста можуть вважатися недоліком. Крім того, не можна запустити BSD, Solaris, OS/X або Windows, в контейнері на базі Linux, а іноді і цей факт також можна вважати недоліком [1,4].

Ідея віртуалізації на рівні процесів саме по собі не нова, а вже була реалізована за допомогою зон Solaris так як і пов'язаних BSD вже кілька років тому. Інші проекти з відкритим вихідним кодом впровадження віртуалізації на рівні процесів існували протягом кількох років. Однак, вони вимагали ядер використання, які часто ставали основною перешкодою. Повна і стабільна підтримка для контейнерів на базі Linux для звичайних ядер в рамках проекту LXC (LinuX Containers) появилася відносно недавно, як можна буде побачити в даній статті. Це робить контейнери більш привабливим для cloud-інфраструктури. Все більше і більше компаній хостингових і cloud-послуг приймають і дотримуються ідей контейнерів на базі Linux. У цій статті *ставиться за мету* описати деякі контейнерні проекти з відкритим кодом на базі Linux і функції ядра, які вони використовують, а також продемонструвати деякі приклади використання. *Метою* також буде опис інструменту Docker для створення LXC контейнерів [4].

Базова інфраструктура сучасних контейнерів на базі Linux складається в основному з двох властивостей ядра: просторів імен і С-груп. Є шість типів просторів імен, які забезпечують для кожного процесу ізоляцію наступних ресурсів операційної системи: файлових систем (MNT), UTS, IPC, PID, просторів імен мережі і користувача (простори імен користувачів дозволяють відображення UID і GID між простором імен користувача і глобальним простором імен хоста). Використовуючи мережеві простори імен, наприклад, кожен процес може мати власний примірник мережевого стека (мережевих інтерфейсів, сокетів, таблиць маршрутизації і правил маршрутизації, правил мережевого фільтра (Netfilter) і так далі).

Створення мережевого простору імен дуже просте і може бути зроблено за допомогою наступної команди `ip route`: `ip netns add myns1`. За допомогою команди `ip netns` також легко переміщати один мережевий інтерфейс з одного мережевого простору імен до іншого для контролю створення і видалення мережевих просторів імен, з метою з'ясувати, якому мережевому простору імен належить визначений процес і так далі. Цілком аналогічно, при використанні простору імен MNT під час монтування файлової системи інші процеси не будуть бачити це монтування, а уже при роботі з просторами імен PID, запустивши на виконання команду ps від даного простору імен PID можна побачити тільки процеси, які були створені виходячи саме з цього простору імен PID.

Підсистема С-груп забезпечує управління ресурсами та їх облік. Це дозволяє легко визначити, наприклад, максимальний обсяг пам'яті використання процесом і може реалізуватися за допомогою VFS-операцій С-груп. Проект С-груп було вперше розпочато двома розробниками Google, Павлом Мінеджі і Рогітом Сетом ще в 2006 році, і він спочатку називався "контейнери процесу" [1]. Ні простори імен, ні С-групи не брали участі в критичних шляхах ядра, і, таким чином, вони не містять значних негативних показників високої продуктивності, за винятком пам'яті для С-групи, яка може включати значні затрати при деяких робочих навантаженнях.

Контейнери на базі Linux

В принципі, контейнер являє собою Linux процес (або декілька процесів), що має свої особливості і працює в ізольованому середовищі, налаштованому на хості. Можна іноді стикатися з такими термінами, як віртуальне середовище (ВС або virtual environment VE) і Віртуальний власний сервер (ВВС або Virtual Private Server VPS) для контейнера.

Особливості цього контейнера залежить від того, як контейнер сконфігурований і на якому базовому Linux-контейнері використовується, так як контейнери Linux реалізовані по-різному в різних проектах. Це обговорюється в найбільш важливих публікаціях.

Походження проекту OpenVZ [2] є власним варіантом реалізації серверів віртуалізації під назвою Virtuozzo, який спочатку був запущений компанією під назвою SWsoft, заснованою в 1997 році. У 2005 році частина продукту Virtuozzo була випущена в якості відкритого вихідного проекту, і він називався OpenVZ. В 2008 році SWsoft об□єдналася з компанією Parallels. OpenVZ

© Мельник В.М., Пех П.А., Мельник М.М.

використовується для надання хостингу і cloud-сервісів, що являється основою спаралелених cloud-серверів. Як Virtuozzo так і OpenVZ базується на модифікованому ядрі Linux. До того ж, у нього є інструменти командного рядка для управління контейнерів, і це дає можливість створювати контейнери з використанням шаблонів для різних дистрибутивів Linux. OpenVZ також може працювати на деяких немодифікованих ядрах, однак зі зменшеним набором функцій. Проект OpenVZ призначений для повноцінного впровадження в майбутньому, але це може зайняти досить багато часу.

В 2013 році компанія Google випустила версію свого власного контейнерного стека з відкритим вихідним кодом lmctfy (що розшифровується як Let Me Contain That For You). На даний час це все ще в стадії тестування. Проект lmctfy заснований на використанні C-груп. На даний час контейнери Google не використовують особливості простору імен ядра, що використовується в інших контейнерних проектах на базі Linux, але використання цієї особливості для контейнерного проекту Google очікується в майбутньому [1].

Проект з відкритим вихідним кодом [3], який був вперше реалізований в 2001 році, забезпечує можливість для безпечного розподілу ресурсів на хості, який, в свою чергу, повинен запустити модифіковане ядро.

Проект LXC (LinuX Containers) надає користувачу набір інструментів і утиліт для управління контейнерами Linux [4]. Багато LXC розробників вийшли з OpenVZ групи. В супереч OpenVZ він працює на немодефікованому ядрі. LXC повністю написаний для контексту користувача і підтримує прив□язки на інших мовах програмування, таких як Python, Lua і Go. Він доступний для більшості популярних дистрибутивів, таких як Fedora, Ubuntu, Debian та інших. Red Hat Enterprise Linux 6 (RHEL6) представив контейнери Linux в якості технічного попереднього перегляду. Можна запустити Linux контейнери на архітектурах, інших ніж x86, таких як ARM архітектури (є кілька прикладів в Інтернеті для запуску контейнерів на Raspberry Pi).

Можна також згадати драйвер Libvirt-LXC, за допомогою якого можна управляти контейнерами [5]. Це робиться шляхом визначення конфігураційних файлів XML, а потім працює `virsh start`, `virsh console` і `visrh destroy` для роботи, розміщення і знищення контейнера, відповідно. Зверніть увагу, що немає єдиного коду між Libvirt-LXC та LXC-проекту користувача.

Управління контейнерами LXC

По-перше, слід переконатися, що хост підтримує LXC, запустивши `LXC-checkconfig` [4]. Якщо все гаразд, можна створити контейнер за допомогою одного з декількох готових шаблонів для створення контейнерів. У LXC-0,9 є таких шаблонів, в основному для популярних дистрибутивів Linux. Можна легко адаптувати ці шаблони відповідно до власних вимог при необхідності. Так, наприклад, можна створити контейнер Fedora під назвою fedoraCT за допомогою команди:

```
lxc-create -t fedora -n fedoraCT
```

Контейнер буде створений за замовчуванням, в `/var/lib/lxc/fedoraCT`. Можна встановити інший шлях для згенерованого контейнера, додавши `--lxcpath` опцію шляху. Опція `-t` визначає ім□я шаблону, який необхідно використовувати (в даному випадку Fedora), і опція `-n` вказує на ім□я контейнера (в даному випадку `fedoraCT`). Слід звернути увагу на те, що можна також створити контейнери інших розподілів на Fedora, наприклад Ubuntu (для нього необхідний пакет Debootstrap). Не всі також комбінації гарантується.

Можна передавати параметри в `lxc-create` після додавання `--`. Наприклад, можна створити старшу версію для кількох дистрибутивів з опціями `-R`, або `-r`, залежно від шаблону дистрибутиву. Для створення старшої версії контейнера Fedora на тому ж хості, Fedora 20, слід вжити команду:

```
lxc-create -t fedora -n fedora19 -- -R 19
```

Можна видалити установку LXC-контейнера з файлової системи за допомогою:

```
lxc-destroy -n fedoraCT
```

Для більшості шаблонів, коли вони використовуються вперше, декілька файлів необхідного пакету завантажуються і кешуються на диску в `/var/cache/lxc`. Ці файли використовуються

© Мельник В.М., Пех П.А., Мельник М.М.

при створенні нового контейнера з тим же самим шаблоном, і, в результаті, створення контейнера, який використовує цей же шаблон, буде швидшим в наступному. Можна почати контейнер, який був створений:

```
lxc-start -n fedoraCT
```

і зупинити з:

```
lxc-stop -n fedoraCT
```

Сигнал, що використовувався `lxc-stop` є SIGPWR за замовчуванням. Для того, щоб використовувати SIGKILL в попередньому прикладі, слід додати `-k` для `lxc-stop`:

```
lxc-stop -n fedoraCT -k
```

Можна також створити контейнер в якості демона, додавши `-d`, а потім увійти в нього з `lxc-console`, як це показано нижче:

```
lxc-start -d -n fedoraCT lxc-console -n fedoraCT
```

Перший `lxc-console`, який можна запустити для даного контейнера зв□яжеться з `tty1`. Якщо `tty1` вже використовується (бо це друга LXC-консоль, яка запущена для цього контейнера), то відбудеться з□єднання з `tty2` і так далі. Слід мати на увазі, що максимальна кількість `tty□s` налаштовується на початку `lxc.tty` конфігураційного файлу контейнера. Можна зробити миттєвий знімок незапущеного контейнера через:

```
lxc-snapshot -n fedoraCT
```

Це дозволить створити миттєвий знімок під `/var/lib/lxcsnaps/fedoraCT`. Перший знімок, що буде створений, буде називатися `snap0`. Другий □ буде називатися `snap1` і так далі. Можна відтворити в часі миттєвий знімок на пізніший час з опцією `-r` □ наприклад:

```
lxc-snapshot -n fedoraCT -r snap0 restoredFdoraCT
```

Можна перерахувати знімки, використовуючи:

```
lxc-snapshot -L -n fedoraCT
```

Можна відобразити запущені контейнери використавши:

```
lxc-ls --active
```

Контейнери управління також можуть бути зроблені за допомогою скриптів, з використанням мов сценаріїв. Наприклад, цей короткий скрипт *Python* запускає контейнер *fedoraCT*:

```
#!/usr/bin/python3 import lxc container = lxc.Container ("fedoraCT")
                   container.start()
```

Конфігурація контейнерів

Файл конфігурації за замовчуванням створюється для кожного новоствореного контейнера в `/var/lib/lxc/<containerName>/config`, але можна змінити це за допомогою опції задавання місця знаходження `--lxcpath` [5]. Можна також налаштувати різні параметри контейнера, наприклад, параметри мережі, параметри С-груп, пристроїв та багато іншого. Наведемо деякі приклади популярних елементів конфігурації для файла конфігурації контейнера:

– Можете встановити різні параметри С-груп шляхом установки значень у входи `lxc.c-group.[subsystem name]` у файлі конфігурації. Ім□я підсистеми є іменем контролера С-групи. Наприклад, конфігуруючи максимальний обсяг пам□яті 256 MB, які контейнер може використовувати, можна через встановлення властивості `lxc.c-group.memory.limit_in_bytes` значення 256.

– Можна налаштувати контейнер хоста, встановивши `lxc.utsname`.

– Є п□ять типів мережевих інтерфейсів, які можна встановити за допомогою параметра `lxc.network.type`: `empty`, `veth`, `vlan`, `macvlan` і `phys`.`veth` дуже часто використовується для можливості підключення контейнера із зовнішнім світом. При використанні `phys` можна переміщати мережеві інтерфейси від мережевого простору імен вузла в простір імен контейнера.

– Існують можливості, які можуть бути використані для покращення безпеки LXC контейнерів. Можна уникнути реалізації деяких зазначених системних викликів зсередини контейнера, встановивши захищений режим обчислень, або `seccomp`, політику входу через `lxc.seccomp` у файлі конфігурації. Також можна відключити ці можливості з контейнера через вхід `lxc.cap.drop`. Наприклад, встановлення `lxc.cap.drop = sys_module` створить контейнер

© Мельник В.М., Пех П.А., Мельник М.М.

без можливості `CAP_SYS_MDOULE`. Спроба запустити insmod зсередини цього контейнера не вдасться. Також для такого контейнера можна визначити профілі `AppArmor` і `SELinux`, а також знайти приклади в `LXC README` і в `man 5 lxc.conf`.

Докер

Docker – це проект з відкритим вихідним кодом, який автоматизує створення і розгортання контейнерів [6]. Перший docker був випущений в березні 2013 року, ліцензія Apache версії 2.0. Вона стартувала в той час як внутрішній проект платформенно-сервісної (Platform-as-a-Service або PaaS) компанії під назвою dotCloud, і тепер називається Docker Inc. Вихідний прототип був написаний в Python, а пізніше весь проект був переписаний в мовою програмування Go, яка була розроблена вперше в Google. У вересні 2013 року, Red Hat оголосила, що вона буде співпрацювати з Docker Inc для Red Hat Enterprise Linux і для платформи Red Hat OpenShift. Для Docker потрібне ядро Linux 3.8 (або вище). У системах RHEL Docker працює під ядро 2.6.32, так як і були перенесені необхідні виправлення.

Docker використовує інструментарій LXC який є в даний час доступний тільки для Linux. Він працює на дистрибутивах, таких як Ubuntu 12.04, 13.04; Fedora 19 і 20; RHEL 6.5 і вищих; і на cloud-платформах, таких як Amazon EC2, Google Compute Engine і Rackspace.

Зображення Docker може зберігатись в загальнодоступному репозиторії і може бути завантажене за допомогою команди `docker pull`, наприклад, `docker pull ubuntu` або `docker pull busybox`. Для відображення зображень наявних на хості можна використовувати команду `docker images`. Можна звузити команду для певного типу зображень (наприклад, Fedora) за допомогою `docker images fedora`. На Fedora, працюючи з Docker-контейнером Fedora, все простіше. Після установки `docker-io package` можна просто запустити `docker dæmon` з `systemctl start docker`, а потім можна стартувати Docker контейнер Fedora як `docker run -i -t fedora /bin/bash`.

Docker має *git*-подібні можливості для обробки контейнерів. Зміни, внесені в контейнері, будуть втрачені, якщо контейнер знищиться і вони не зафіксуються (так само, як робиться в *git*) з `docker commit <containerId> <containerName/containerTag>`. Ці зображення можуть бути завантажені на доступний регістр, і відкриті для бажаючих завантажити їх. Крім того, можна встановити власне ззовні недоступнене сховище Docker.

Docker здатний створити знімок за допомогою можливостей пристрою зображення ядра. У більш ранніх версіях до версії Докер 0.7 це було зроблено за допомогою AUFS (union filesystem). Docker 0.7 складає "плагіни зберігання" для того, щоб при необхідності можна було перемикатися між відображеннями пристроїв і AUFS (якщо ядро підтримує його). Таким чином Docker може працювати на релізи RHEL, які не підтримують AUFS.

Можна створювати зображення, виконуючи команди вручну і створювати в результаті контейнер, але також можна описати їх з Dockerfile. Так само як Makefile буде компілювати код в двійковий файл виконання, Dockerfile будуватиме готове до запуску контейнерне зображення виходячи з простих інструкцій. Команда для створення зображення з Dockerfile є `docker build`. Існує підручник про Dockerfiles і їх синтаксис команд на веб-сайті Docker. Наприклад, після короткого Dockerfile для установки пакета `iperf` для зображення Fedora:

```
FROM fedora MAINTAINER Rami Rosen RUN yum install -y iperf
```

Можна завантажувати і зберігати зображення безкоштовно на відкритому індексі Докера. Так само, як і у випадку з GitHub, зберігання доступних зображень безкоштовне і просто вимагає, щоб користувач зареєструвався.

Можливість КТВ

Проект КТВ (контрольна точка / відновлення) реалізується в основному в просторі користувача і існує більш ніж 100 маленьких ″ділянок″, розміщені по ядру для його підтримки. Було кілька спроб реалізації КТВ окремо в просторі ядра, а деякі з них – в рамках проекту OpenVZ. Спільнота ядра відхилила всі з них через те, що вони були занадто складні [7].

Можливість КТВ дозволяє зберегти стан процесу в декількох файлах зображень і відновити цей процес з точки, в якій він був остановлений, на тому ж хості або на іншому пізніше в часі. Цей процес також може бути контейнером LXC. Файли зображень створюються з використанням

© Мельник В.М., Пех П.А., Мельник М.М.

формату буфера протоколу (БП) Google. Можливість КТВ дозволяє виконання завдань, таких як оновлення технічного обслуговування ядра або апаратного обладнання на цьому ж хості після моменту перевірки його додатків для постійного зберігання. Пізніше, додатки будуть відновлені на цьому хості.

Ще одною дуже важливою особливістю є балансування навантаження за допомогою динамічної міграції. Можливості КТ/В також можуть бути використані для створення інкрементних знімків, які можуть бути використані після виникнення аварії. Як уже згадувалося раніше, деякі "ділянки" ядра були необхідні для підтримки КТ/В в просторі користувача. Наведемо деякі з них:

- Був доданий новий системний виклик `kcmp()` який порівнює два процеси, щоб визначити, чи вони займають ресурс ядра.
- Інтерфейс сокетного моніторингу `sock_diag` був доданий в сокети UNIX щоб бути в змозі знайти знаходження сокета UNIX домена. Перед цією зміною інструмент `ss`, який спирався на аналіз записів `/proc`, не виявив цю інформацію.
- Був доданий режим налагодження TCP з єднання.
- Був доданий вхід `procfs` (/proc/PID/map_files).
- Давайте подивимося на простий приклад використання функції `criu` (КТ/В в просторі користувача). По-перше, необхідно перевірити, чи підтримує ядро КТ/В, запустивши `criu check --ms`. Слід завбачити відповідь, яка говорить `"Looks good."`

В основному, визначена перевірка здійснюється через:

```
criu dump -t <pid>
```

Можна вказати папку, в якій наявні файли процесу будуть збережені, додавши `-D folderName` і можна відновити їх з використанням `criu restore <pid>`.

Висновки

У даній роботі описано контейнери на базі Linux і коротко пояснено, що в основі лежать С-групи і простір імен функцій ядра. Обговорено кілька проектів з використанням контейнерів на базі Linux, з концентрацією уваги на перспективний і популярний проект LXC. Також в роботі приділено увагу `Docker engine` наоснові LXC, який забезпечує легкий і зручний спосіб для створення і розширення LXC-контейнерів. На прикладі кількох наочних прикладів показано, як просто конфігурувати LXC-контейнери, керувати ними і розгортати їх з використанням засобів LXC простору користувача та засобів Docker.

Використовуючи переваги LXC- і Docker-проектів з відкритим кодом та зручні і прості інструменти для створення, розгортання і налаштування LXC-контейнерів, як описано в цій статті, можна буде, ймовірно, побачити більше і більше cloud-інфраструктур, які будуть інтегрувати LXC контейнери замість використання віртуальних машин в найближчому майбутньому. Тим не менш, як наголошується в цій роботі, рішення, такі як Xen або KVM мають ряд переваг перед контейнерами на базі Linux і все ж є необхідними, так що вони, ймовірно, не можуть віджити з cloud-інфраструктури в найближчі кілька років.

1. Google Containers: https://github.com/google/lmctfy
2. OpenVZ: http://openvz.org/Main_Page
3. Linux-VServer: http://linux-vserver.org/
4. LXC: http://linuxcontainers.org/
5. libvirt-lxc: http://libvirt.org/drvlxc.html
6. Docker: https://www.docker.io/
7. Docker Public Registry: https://index.docker.io/

© Мельник В.М., Пех П.А., Мельник М.М.