Кластеризация + виртуализация : Linux HA + OpenVZ

Евгений Прокопьев

28 мая 2012 г.

Содержание

Аннотация

Как одновременно уменьшить количество физических серверов, требующих обслуживания, и при этом продублировать их, обеспечив автоматическое переключение на резервный сервер при отказе основного

Чем больше сервисов находится на попечении системного администратора (или нескольких администраторов) и чем в большей степени от их работоспособности зависит нормальное функционирование предприятия, тем более актуальными становятся следующие задачи:

- *Отказоустойчивость* — от простого резервного копирования важных данных до систем высокой готовности (High Availability), когда при выходе из строя основного сервера его функции автоматически берет на себя резервный сервер.
- *Консолидация сервисов и сокращение количества физических серверов, требующих обслуживания* — от простого размещения нескольких сервисов на одном физическом сервере до использования различных систем виртуализации, изолирующих сервисы друг от друга.

На первый взгляд эти задачи кажутся прямо прямо противоположными, однако их вполне можно совместить, если обеспечить отказоустойчивость не каждого сервиса в отдельности, а *группы сервисов*, работающих в виртуальных средах.

1 Инструменты

Сначала определимся с требуемым инструментарием, исходя из того, что нас интересуют только решения, целиком построенные из свободного программного обеспечения.

Если требуется гарантировать отказоустойчивость сервису, который сам по себе отказоустойчивым не является и встроенных механизмов кластеризации не имеет, то особых альтернатив проекту Linux HA (`http://linux-ha.org`) нет. Основным компонентом проекта является менеджер кластера Heartbeat, предоставляющий средства коммуникации между узлами кластера (в том числе механизм оповещения об отключении/включении узла) и управления ресурсами кластера. Для сервисов, решающих исключительно вычислительные задачи, этого может быть достаточно, но большинство сервисов имеют привычку также хранить свои

настройки и данные на диске. Чтобы они были доступны сервису независимо от того, на каком узле кластера он в данный момент работает, требуются внешние по отношению к проекту Linux HA средства организации распределенного файлового хранилища. В их качестве могут выступать:

- *классические распределенные файловые системы (OCFS, CODA, GFS, Lustre)*, позволяющие организовать параллельный доступ к файлам с нескольких компьютеров одновременно;
- *средства автоматического зеркалирования разделов жесткого диска на двух компьютерах*, обеспечивающие доступ к файлам на чтение и запись только с одного компьютера (программные — DRBD и аппаратные — IBM ServeRAID);
- *прочие средства*, которые в принципе использовать можно, но либо их надежность (NFS, CIFS), либо стоимость (SAN) в данном случае являются неприемлемыми.

Чаще всего в связке с Heartbeat используется DRBD (`http://drbd.org`), которая в версии 7 обеспечивает более высокую производительность и надежность по сравнению с классическими распределенными файловыми системами. DRBD версии 8 движется в сторону своих конкурентов, пытаясь сохранить собственные преимущества, но к промышленной эксплуатации она пока еще не готова.

Что же касается виртуализации, то здесь выбор гораздо шире, и определяется он тем, что именно мы собираемся виртуализировать, и сколько аппаратных ресурсов мы готовы потратить. Поскольку нам не требуется запуск в разных виртуальных средах различных операционных систем, разумно будет в целях экономии аппаратных ресурсов остановиться на технологиях виртуализации на уровне операционной системы. Из трех открытых проектов такого рода, предназначенных для Linux, наиболее функциональным и быстрее всего развивающимся выглядит OpenVZ, который также является основой для проприетарного продукта Virtuozzo фирмы SWSoft. Более того, на wiki проекта довольно коротко уже описана удачная попытка «скрестить» его с проектом Linux HA — `http://wiki.openvz.org/HA_cluster_with_DRBD_and_Heartbeat`. Наконец, OpenVZ работает в ALT Linux Sisyphus «из коробки». Поэтому пойдем уже проторенной дорогой и попытаемся построить аналогичное решение, использующее все преимущества кластеризации.

2 Что нам предстоит сделать

В качестве простого примера возьмем ситуацию, когда нам требуется сконфигурировать почтовый сервер и сервер БД. Возможны следующие подходы к решению данной задачи:

1. *Традиционный* — для каждой задачи выделяется и конфигурируется отдельный физический сервер. В случае сбоя одного из серверов мы лишемся как минимум тех сервисов, которые этот сервер предоставляет. В худшем случае, учитывая, что сервисы могут зависеть друг от друга (а конфигурации, в которых почтовый сервер интенсивно использует сервер БД, не так уж редки), мы можем лишиться гораздо большего.

2. *Кластеризация* — физические сервера (точнее, сервисы, работающие на этих серверах) дублируются средствами Linux HA. В этом случае надежность значительно увеличивается, но возрастает также количество серверов и затраты на их обслуживание.

3. *Виртуализация* — для каждого сервера выделяется отдельная виртуальная среда на общем физическом сервере. В этом случае увеличивается гибкость, теперь мы не ограничены в количестве серверов и можем, например, выделить отдельную виртуальную среду для организации маршрутизатора/брандмауэра, ограничивающего доступ к почтовому серверу и серверу БД. Количество физических серверов и затраты на их обслуживание уменьшаются, но вместе с ними уменьшается и надежность, которая может оказаться даже ниже, чем в случае (1), если работоспособность серверов зависит друг от друга.

4. *Кластеризация + виртуализация* — общий физический сервер (точнее, один единственный сервис — система виртуализации) дублируется средствами Linux HA, тем самым мы совмещаем преимущества подходов (2) и (3) ценой увеличения нагрузки на сервера по сравнению с подходом (1).

Если не углубляться в детали, схема выбранного нами последнего способа построения системы изображена на рисунке 1.

3 Строим кластер

Более подробная схема кластера, который нам необходимо построить, изображена на рисунке 2. Каждый узел кластера имеет по 3 сетевые

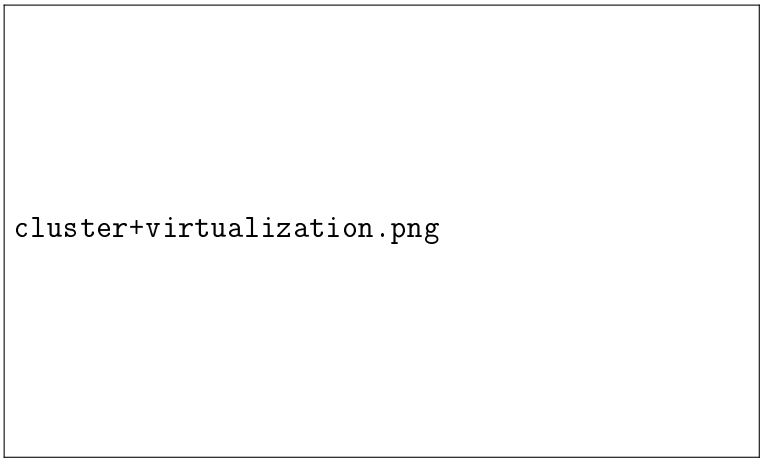

Рис. 1: Кластеризация + виртуализация

карты:

- `eth0` используется для подключения к внешней сети;
- `eth1` будет использована далее при создании виртуального сервера router, поэтому IP-адрес ей не присвоен;
- `eth2` специально выделена для коммуникации между узлами кластера.

3.1 Синхронизированные разделы: DRBD

Сначала настроим общие разделы узлов кластера средствами DRBD. Установка DRBD производится штатным для ALT Linux способом. Смотрим, какие пакеты имеются в наличии:

```
[root@m1 ~]# apt-cache search drbd
drbd-tools - Distributed Redundant Block Device utilities
kernel-modules-drbd-ovz-smp - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-std-smp - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-std-up - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-std26-smp - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-std26-up - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-vs26-smp - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-wks26-smp - Linux drbd kernel modules for DRBD.
kernel-modules-drbd-wks26-up - Linux drbd kernel modules for DRBD.
kernel-source-drbd-0.7.21 - Kernel source for DRBD
```

Рис. 2: Схема кластера

В репозиториях ALT Linux ядро имеется в скомпилированном и упакованном в RPM-пакеты виде в нескольких вариантах. Каждый вариант оптимизирован под свою схему применения Такие пакеты называются по стандартной схеме `kernel-image-тип-сборки`.

Практика ALT Linux показывает, что пользователю никогда не требуется самостоятельно компилировать ядро, достаточно выбрать один из имеющихся вариантов.

Часть модулей ядра не включается в общий пакет с образом ядра, они распространяются в отдельных пакетах (`kernel-modules-тип-сборки`), что позволяет пользователю устанавливать и удалять их независимо. Однако при этом модули должны быть согласованы с ядром по версии и по типу сборки.

Затем определяем, какое у нас ядро:

```
[root@m1 ~]# uname -a
Linux m1.mydomain.com 2.6.16-std26-up-alt10 #1 Wed Sep 13 20:06:02 MSD 2006 i686
```

На основании этой информации (`std26-up`) решаем, какие именно пакеты нам нужны, и устанавливаем их:

```
[root@m1 ~]# apt-get install kernel-modules-drbd-std26-up drbd-tools
```

Первый установленный нами пакет содержит модуль ядра, необходимый для работы DRBD, второй — userspace-инструменты для управления DRBD. Конфигурирование DRBD сводится к редактированию файла /etc/drbd.conf. Минимальная, но вполне работоспособная конфигурация будет выглядеть так:

```
resource r0 {

    # протокол передачи
    # C: запись считается завершенной, если данные записаны на локальный и удале
    #    подходит для критически важных транзакций и в большинстве остальных слу
    # B: запись считается завершенной, если данные записаны на локальный диск
    #    и удаленный буферный кэш
    # A: запись считается завершенной, если данные записаны на локальный диск
    #    и локальный буфер tcp для отправки
    #    подходит для для сетей с высокой задержкой
    protocol B;

    # описание узла m1
    on m1.mydomain.com {
        device     /dev/drbd0;           # имя drbd-устройства
        disk       /dev/hda3;            # раздел диска, поверх которого оно раб
        address    192.168.200.1:7788;   # адрес и порт демона drbd
        meta-disk  internal;             # хранить метаданные drbd на том же раз
    }

    # описание узла m2
    on m2.mydomain.com {
        device    /dev/drbd0;            # имя drbd-устройства
        disk      /dev/hda3;             # раздел диска, поверх которого оно раб
        address   192.168.200.2:7788;    # адрес и порт демона drbd
        meta-disk internal;              # хранить метаданные drbd на том же раз
    }
}
```

В комплекте с DRBD поставляется очень детальный пример конфигурационного файла, в котором все параметры прокомментированы.

Для работы DRBD необходим специальный файл устройста (/dev/drbd0). Если в системе используется udev, то такой файл будет создаваться автоматически, в противном случае придется создать его вручную:

```
[root@m1 ~]# mknod /dev/drbd0 b 147 0
```

Последний шаг настройки — обеспечить запуск сервиса drbd при загрузке узла:

```
[root@m1 ~]# chkconfig --level 2345 drbd on
```

Эти операции необходимо повторить на узле m2, а затем на обоих узлах запустить сервис drbd:

```
[root@m1 ~]# service drbd start
Starting Starting DRBD resources:     service: [ d0 s0 n0 ].
...
```

После чего на обоих узлах кластера мы должны увидеть следующее:

```
[root@m1 ~]# service drbd status
drbd driver loaded OK; device status:
version: 0.7.21 (api:79/proto:74)
SVN Revision: 2326 build by builder@xeon.office.altlinux.ru, 2006-09-22 23:46:26
 0: cs:Connected st:Secondary/Secondary ld:Inconsistent
    ns:0 nr:0 dw:0 dr:0 al:0 bm:0 lo:0 pe:0 ua:0 ap:0
```

Итак, устройство /dev/drbd0 уже работает, но содержимое разделов /dev/hda3, лежащих уровнем ниже, еще не синхронизировано. Первый раз синхронизацию необходимо произвести вручную, выполнив на узле m1:

```
[root@m1 ~]# drbdadm -- --do-what-I-say primary all
```

После выполнения этой команды мы можем проследить за процесом синхронизации:

```
[root@m1 ~]# service drbd status
drbd driver loaded OK; device status:
version: 0.7.21 (api:79/proto:74)
SVN Revision: 2326 build by builder@xeon.office.altlinux.ru, 2006-09-22 23:46:26
 0: cs:SyncSource st:Primary/Secondary ld:Consistent
    ns:1972 nr:0 dw:0 dr:10164 al:0 bm:0 lo:0 pe:30 ua:2048 ap:0
       [>..................] sync'ed:  0.2% (3158696/3160552)K
       finish: 0:26:19 speed: 1,856 (1,856) K/sec
```

После окончания синхронизации на узле m1 мы увидим:

```
[root@m1 ~]# service drbd status
drbd driver loaded OK; device status:
version: 0.7.21 (api:79/proto:74)
SVN Revision: 2326 build by builder@xeon.office.altlinux.ru, 2006-09-22 23:39:33
 0: cs:Connected st:Primary/Secondary ld:Consistent
    ns:731068 nr:1052796 dw:1782732 dr:193337 al:20 bm:356 lo:0 pe:0 ua:0 ap:0
```

По строке Primary/Secondary можно определить, что сейчас узел m1 является ведущим. На ведомом узле m2 в выводе той же самой команды мы увидим Secondary/Primary. Все операции с устройством /dev/drbd0 необходимо выполнять только на ведущем узле, при этом все изменения будут автоматически применены к разделам /dev/hda3 на обоих узлах кластера.

Создадим на устройстве /dev/drbd0 файловую систему и смонтируем ее:

```
[root@m1 ~]# mkfs.ext3 /dev/drbd0
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
395200 inodes, 790138 blocks
39506 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=809500672
25 block groups
32768 blocks per group, 32768 fragments per group
15808 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 25 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@m1 ~]# mkdir /d0
[root@m1 ~]# mount /dev/drbd0 /d0
```

Если теперь мы попытаемся смонтировать устройство /dev/drbd0 на узле m2, мы получим следующее сообщение об ошибке:

```
[root@m2 ~]# mkdir /d0
[root@m2 ~]# mount /dev/drbd0 /d0
/dev/drbd0: Input/output error
mount: block device /dev/drbd0 is write-protected, mounting read-only
/dev/drbd0: Input/output error
mount: /dev/drbd0 already mounted or /d0 busy
```

Такое (вполне разумное) поведение гарантируется только для ядер 2.6, в 2.4 была возможность смонтировать устройство /dev/drbd0 на обоих узлах кластера одновременно, что, в свою очередь, могло привести к повреждению файловой системы.

Чтобы все-таки смонтировать раздел /dev/drbd0 на узле m2, необходимо сделать его ведущим, выполнив на узлах m1 и m2 следующие команды:

```
[root@m1 ~]# umount /d0
[root@m1 ~]# drbdadm disconnect all
[root@m2 ~]# drbdadm disconnect all
[root@m1 ~]# drbdadm secondary all
[root@m2 ~]# drbdadm -- --human primary all
[root@m1 ~]# drbdadm connect all
[root@m2 ~]# drbdadm connect all
[root@m2 ~]# mount /dev/drbd0 /d0
```

В случае отказа узла m1 достаточно выполнить только те команды, которые приведены выше для узла m2. По большому счету, любители велосипедостроения (или оптимальных решений — иногда трудно отличить одно от другого) могут этим и ограничиться: написать скрипт, который будет следить на доступностью соседнего узла и монтировать раздел /dev/drbd0, если соседний узел перестал отвечать, можно самостоятельно. Другой вопрос: зачем, если Heartbeat умеет делать и это, и многое другое?

3.2 Автоматическое отслеживание узлов: Heartbeat

Устанавливаем Heartbeat штатным для ALT Linux способом:

```
[root@m1 heartbeat]# apt-get install heartbeat
```

Установка Heartbeat для других дистрибутивов Linux описана на сайте проекта (`http://www.linux-ha.org/DownloadSoftware`). На главной странице проекта также сказано, что кроме Linux Heartbeat собирается и запускается на FreeBSD, Solaris и даже MacOS X.

После установки Heartbeat на оба узла кластера на каждом узле необходимо создать файл /etc/ha.d/authkeys с правами доступа 600 и случайной строкой (ее можно придумать самому, а лучше сгенерировать с помощью apg), которая будет использоваться узлами для авторизации друг друга:

```
auth 1
1 sha1 RcBkJzU8ClnrjWVRLv5EDsdRFQP1j1C
```

Также необходимо создать конфигурационный файл Heartbeat, который на узле m1 будет выглядеть так:

```
Logfacility     local0
ucast eth2 192.168.200.2
auto_failback on
node m1.mydomain.com m2.mydomain.com
```

а на m2 — так:

```
Logfacility     local0
ucast eth2 192.168.200.1
auto_failback on
node m1.mydomain.com m2.mydomain.com
```

Затем необходимо описать ресурсы кластера в файле /etc/ha.d/haresources на каждом узле:

```
m1.mydomain.com drbddisk Filesystem::/dev/drbd0::/d0::ext3
```

Такая запись означает, что кластер в штатном режиме будет использовать ресурсы drbddisk и Filesystem узла m1, а в случае его «смерти» — аналогичные ресурсы узла, оставшегося в живых, то есть m2. Для ресурса Filesystem заданы параметры: имя drbd-устройства, каталог, в который оно должно быть смонтировано (этот каталог мы должны создать самостоятельно на каждом узле кластера), и тип файловой системы. Узнать больше о ресурсах, поддерживаемых heartbeat, можно заглянув в каталог /etc/ha.d/resource.d — каждый ресурс представлен там соответствующим скриптом.

Теперь можно запустить сервис heartbeat (не забыв перед этим размонтировать устройство /dev/drbd0, если оно было смонтировано):

```
[root@m1 ~]# service heartbeat start
logd is already running
Starting High-Availability services:       [ DONE ]
```

После старта сервиса на ведущем узле кластера в логах можно увидеть такие сообщения:

```
m1 heartbeat: [3372]: info: Status update for node m2.mydomain.com: status activ
m1 harc[3395]: info: Running /etc/ha.d/rc.d/status status
m1 heartbeat: [3406]: info: Local Resource acquisition completed.
m1 harc[3431]: info: Running /etc/ha.d/rc.d/ip-request-resp ip-request-resp
m1 ip-request-resp[3431]: received ip-request-resp drbddisk OK yes
m1 ResourceManager[3446]: info: Acquiring resource group: m1.mydomain.com drbddi
m1 ResourceManager[3446]: info: Running /etc/ha.d/resource.d/drbddisk  start
m1 Filesystem[3569]: INFO: Running status for /dev/drbd0 on /d0
m1 Filesystem[3569]: INFO: /d0 is unmounted (stopped)
m1 Filesystem[3505]: INFO: Filesystem Resource is stopped
m1 ResourceManager[3446]: info: Running /etc/ha.d/resource.d/Filesystem /dev/drb
m1 Filesystem[3678]: INFO: Running start for /dev/drbd0 on /d0
m1 kernel: kjournald starting.  Commit interval 5 seconds
m1 kernel: EXT3 FS on drbd0, internal journal
m1 kernel: EXT3-fs: mounted filesystem with ordered data mode.
m1 Filesystem[3614]: INFO: Filesystem Success
```

В логах ведомого узла можно будет увидеть следующее:

```
m2 heartbeat: [3739]: info: Status update for node m1.mydomain.com: status up
m2 harc[3752]: info: Running /etc/ha.d/rc.d/status status
m2 heartbeat: [3739]: info: remote resource transition completed.
m2 heartbeat: [3773]: info: No local resources [/usr/lib/heartbeat/ResourceManag
```

После старта heartbeat на обоих узлах кластера устройство /dev/drbd0 будет смонтировано на ведущем узле. Если остановить сервис heartbeat на ведущем узле m1, ведомый m2 возьмет на себя функции ведущего, и устройство /dev/drbd0 будет смонтировано на нем, при этом в логах мы увидим:

```
m2 kernel: drbd0: Secondary/Primary --> Secondary/Secondary
m2 heartbeat: [3739]: info: Received shutdown notice from 'm1.mydomain.com'.
m2 heartbeat: [3739]: info: Resources being acquired from m1.mydomain.com.
m2 heartbeat: [3850]: info: acquire local HA resources (standby).
m2 heartbeat: [3851]: info: No local resources [/usr/lib/heartbeat/ResourceManag
```

```
m2 heartbeat: [3850]: info: local HA resource acquisition completed (standby).
m2 heartbeat: [3739]: info: Standby resource acquisition done [all].
m2 harc[3870]: info: Running /etc/ha.d/rc.d/status status
m2 mach_down[3880]: info: Taking over resource group drbddisk
m2 ResourceManager[3900]: info: Acquiring resource group: m1.mydomain.com drbddi
m2 ResourceManager[3900]: info: Running /etc/ha.d/resource.d/drbddisk start
m2 kernel: drbd0: Secondary/Secondary --> Primary/Secondary
m2 Filesystem[4023]: INFO: Running status for /dev/drbd0 on /d0
m2 Filesystem[4023]: INFO: /d0 is unmounted (stopped)
m2 Filesystem[3959]: INFO: Filesystem Resource is stopped
m2 ResourceManager[3900]: info: Running /etc/ha.d/resource.d/Filesystem /dev/drb
m2 Filesystem[4132]: INFO: Running start for /dev/drbd0 on /d0
m2 kernel: kjournald starting.  Commit interval 5 seconds
m2 kernel: EXT3 FS on drbd0, internal journal
m2 kernel: EXT3-fs: mounted filesystem with ordered data mode.
m2 Filesystem[4068]: INFO: Filesystem Success
m2 mach_down[3880]: info: /usr/lib/heartbeat/mach_down: nice_failback: foreign r
m2 mach_down[3880]: info: mach_down takeover complete for node m1.mydomain.com.
m2 heartbeat: [3739]: info: mach_down takeover complete.
```

Если просто выключить питание узла m1, чтобы он не успел оповестить m2 о том, что он завершает свою работу, m2 обнаружит отсутствие m1 и точно так же возьмет на себя функции ведущего узла и примонтирует раздел /dev/drbd0:

```
m2 kernel: drbd0: PingAck did not arrive in time.
m2 kernel: drbd0: drbd0_asender [3566]: cstate Connected --> NetworkFailure
m2 kernel: drbd0: asender terminated
m2 kernel: drbd0: drbd0_receiver [3510]: cstate NetworkFailure --> BrokenPipe
m2 kernel: drbd0: short read expecting header on sock: r=-512
m2 kernel: drbd0: worker terminated
m2 kernel: drbd0: drbd0_receiver [3510]: cstate BrokenPipe --> Unconnected
m2 kernel: drbd0: Connection lost.
m2 kernel: drbd0: drbd0_receiver [3510]: cstate Unconnected --> WFConnection
m2 heartbeat: [3739]: WARN: node m1.mydomain.com: is dead
m2 heartbeat: [3739]: WARN: No STONITH device configured.
m2 heartbeat: [3739]: WARN: Shared disks are not protected.
m2 heartbeat: [3739]: info: Resources being acquired from m1.mydomain.com.
m2 heartbeat: [3739]: info: Link m1.mydomain.com:eth2 dead.
m2 harc[4383]: info: Running /etc/ha.d/rc.d/status status
m2 heartbeat: [4384]: info: No local resources [/usr/lib/heartbeat/ResourceManag
```

```
m2 mach_down[4403]: info: Taking over resource group drbddisk
m2 ResourceManager[4423]: info: Acquiring resource group: m1.mydomain.com drbddi
m2 ResourceManager[4423]: info: Running /etc/ha.d/resource.d/drbddisk  start
m2 kernel: drbd0: Secondary/Unknown --> Primary/Unknown
m2 Filesystem[4546]: INFO: Running status for /dev/drbd0 on /d0
m2 Filesystem[4546]: INFO: /d0 is unmounted (stopped)
m2 Filesystem[4482]: INFO: Filesystem Resource is stopped
m2 ResourceManager[4423]: info: Running /etc/ha.d/resource.d/Filesystem /dev/drb
m2 Filesystem[4655]: INFO: Running start for /dev/drbd0 on /d0
m2 kernel: kjournald starting.  Commit interval 5 seconds
m2 kernel: EXT3 FS on drbd0, internal journal
m2 kernel: EXT3-fs: recovery complete.
m2 kernel: EXT3-fs: mounted filesystem with ordered data mode.
m2 Filesystem[4591]: INFO: Filesystem Success
m2 mach_down[4403]: info: /usr/lib/heartbeat/mach_down: nice_failback: foreign r
m2 mach_down[4403]: info: mach_down takeover complete for node m1.mydomain.com.
m2 heartbeat: [3739]: info: mach_down takeover complete.
```

При повторном запуске сервиса heartbeat на m1 или при включении узла m1 он снова станет ведущим.

Итак, мы построили отказоустойчивый кластер с общим дисковым пространством, состоящий из двух узлов: m1 и m2. Теперь нам необходимо построить систему виртуализации с виртуальными серверами, мигрирующими с узла m1 на m2 в случае отказа первого.

4 Строим контейнер для виртуальных серверов и кластеризуем его

Традиционно после конфигурирования общего файлового хранилища файл /etc/ha.d/haresources дополняется списком сервисов, для которых необходимо гарантировать отказоустойчивость, а конфигурационные файлы и файлы данных этих сервисов перемещаются на файловую систему, созданную поверх drbd-устройства. Часто также конфигурируется алиас для сетевого интерфейса, который автоматически создается на ведущем узле — тем самым все сервисы оказываются доступными по одному адресу.

Недостатки такого подхода очевидны — чем больше сервисов, тем более запутанной становится такая конфигурация, а установка новых версий сервисов становится очень неудобной. Про общие проблемы исполь-

зования различных сервисов на одном сервере я и не говорю — это специфично не только для Linux HA.

Самым простым и изящным выходом в данной ситуации является виртуализация. Если использовать этот подход, нам потребуется обеспечить отказоустойчивость только для одного сервиса — контейнера, в котором будут жить виртуальные сервера, причем последние можно конфигурировать, не задумываясь о сложностях, связанных с кластеризацией.

Приступим к установке выбранной нами системы виртуализации — OpenVZ. Она представляет собой набор патчей к стандартному ядру Linux и userspace-инструменты для управления. В ALT Linux Sisyphus ядро с поддержкой OpenVZ собрано отдельно, поэтому его необходимо установить в дополнение к существующему ядру, установить модуль поддержки DRBD в этом ядре и userspace-инструменты, представленные пакетами vzctl и vzquota:

```
[root@m1 ~]# apt-get install kernel-image-ovz-smp kernel-modules-drbd-ovz-smp vz
```

После установки нового ядра необходимо проверить конфигурацию загрузчика и удостовериться в том, что после перезапуска будет загружено ovz-ядро, а если это не так, предпринять для загрузки требуемого ядра необходимые действия. Например, если в качестве загрузчика используется lilo, то его конфигурационный файл может выглядеть так:

```
boot=/dev/hda
map=/boot/map
default=linux
image=/boot/vmlinuz
    label=linux
    root=/dev/hda2
    initrd=/boot/initrd.img
    read-only
```

При этом /boot/vmlinuz и /boot/initrd.img должны ссылаться на образ ядра и образ initrd соответственно:

```
[root@m1 ~]# ls -l /boot/
total 6892
-rw-r--r-- 1 root root  755493 Sep 22 22:45 System.map-2.6.16-ovz-smp-alt7
-rw-r--r-- 1 root root  686841 Oct  9 23:54 System.map-2.6.16-std26-up-alt10
-rw-r--r-- 1 root root     512 Oct  9 23:54 boot.0300
-rw-r--r-- 1 root root     512 Oct  9 23:54 boot.0800
```

```
-rw-r--r-- 1 root root   65929 Sep 22 22:39 config-2.6.16-ovz-smp-alt7
-rw-r--r-- 1 root root   66100 Oct  9 23:54 config-2.6.16-std26-up-alt10
-rw------- 1 root root  321240 Oct 10 23:34 initrd-2.6.16-ovz-smp-alt7.img
-rw------- 1 root root  204992 Oct  9 23:54 initrd-2.6.16-std26-up-alt10.img
lrwxrwxrwx 1 root root      30 Oct 10 23:34 initrd-smp.img -> initrd-2.6.16-ovz-
lrwxrwxrwx 1 root root      32 Oct  9 23:54 initrd-up.img -> initrd-2.6.16-std26
lrwxrwxrwx 1 root root      30 Oct 10 23:34 initrd.img -> initrd-2.6.16-ovz-smp-
-rw------- 1 root root   31744 Oct 10 23:38 map
lrwxrwxrwx 1 root root      27 Oct 10 23:34 vmlinuz -> vmlinuz-2.6.16-ovz-smp-al
-rw-r--r-- 1 root root 1246789 Sep 22 22:45 vmlinuz-2.6.16-ovz-smp-alt7
-rw-r--r-- 1 root root 1132834 Oct  9 23:54 vmlinuz-2.6.16-std26-up-alt10
lrwxrwxrwx 1 root root      27 Oct 10 23:34 vmlinuz-smp -> vmlinuz-2.6.16-ovz-sm
lrwxrwxrwx 1 root root      29 Oct  9 23:54 vmlinuz-up -> vmlinuz-2.6.16-std26-u
```

Также с помощью chkconfig –list необходимо удостовериться, что сервис vz не будет запущен после перезагрузки, и затем перезагрузиться.

После перезагрузки необходимо переместить файлы OpenVZ в каталог /d0, куда уже должно быть смонтировано устройство /dev/drbd0, а на старом месте создать символьные ссылки:

```
[root@m1 ~]# mkdir /d0/vz
[root@m1 ~]# mkdir /d0/vz/etc
[root@m1 ~]# mkdir /d0/vz/etc/sysconfig
[root@m1 ~]# mkdir /d0/vz/var
[root@m1 ~]# mkdir /d0/vz/var/lib
[root@m1 ~]# cp -r /etc/vz /d0/vz/etc
[root@m1 ~]# cp -r /etc/sysconfig/vz-scripts /d0/vz/etc/sysconfig
[root@m1 ~]# cp -r /var/lib/vz /d0/vz/var/lib
[root@m1 ~]# cp -r /var/lib/vzquota /d0/vz/var/lib
[root@m1 ~]# rm -rf /etc/vz
[root@m1 ~]# rm -rf /etc/sysconfig/vz-scripts
[root@m1 ~]# rm -rf /var/lib/vz
[root@m1 ~]# rm -rf /var/lib/vzquota
[root@m1 ~]# ln -s /d0/vz/etc/vz /etc/vz
[root@m1 ~]# ln -s /d0/vz/etc/sysconfig/vz-scripts /etc/sysconfig/vz-scripts
[root@m1 ~]# ln -s /d0/vz/var/lib/vz /var/lib/vz
[root@m1 ~]# ln -s /d0/vz/var/lib/vzquota /var/lib/vzquota
```

После остановки сервиса heartbeat на узле m1 и монтирования drbd-устройства на узле m2 на нем необходимо аналогичным образом удалить каталоги OpenVZ и создать вместо них ссылки на /d0/vz.

После того, как OpenVZ перенесен на drbd-раздел, необходимо указать сервису heartbeat, что сервис vz должен работать на узле m1, для чего отредактировать файл /etc/ha.d/haresources на обоих узлах:

```
m1.mydomain.com drbddisk Filesystem::/dev/drbd0::/d0::ext3 vz
```

После перезапуска heartbeat на обоих узлах необходимо смоделировать отказ узла m1 и убедиться в том, что сервис vz запускается на узле m2.

5 Строим первый виртуальный сервер

Теперь мы можем забыть о том, что OpenVZ работает в кластере, и конфигурировать его, опираясь на оригинальную дукументацию с `http://openvz.org/documentation` и выжимку из нее, ориентированную на ALT Linux и доступную здесь — `http://www.freesource.info/wiki/AltLinux/Dokumentacija/OpenVZ`.

Существуют уже готовые шаблоны виртуальных серверов, построенные на основе некоторых общедоступных дистрибутивов Linux: CentOS, Debian, Fedora Core, Gentoo, Mandriva, openSUSE и ALT Linux. Коммерческие RHEL и SUSE SLES в этом списке отсутствуют.

Но мы построим шаблон для виртуального сервера на основе ALT Linux самостоятельно с помощью spt. Если spt уже установлен (его удобнее держать на выделенном физическом либо виртуальном сборочном сервере), то можно использовать содержимое каталога /usr/share/spt/profile-ovz/ как образец для создания образа, который затем послужит нам шаблоном для создания виртуального сервера. Нет никаких препятствий к тому, чтобы использовать этот образец как есть, но мне показалось более правильным скопировать его в /ovz и изменить список пакетов в шаблоне, отредактировав файл /ovz/packages/main так:

```
basesystem
passwd
apt
apt-conf-sisyphus
etcnet
glibc
sysklogd
mc
openssh-server
openssh-clients
```

Также мне показалось разумным изменить конфигурацию apt по умолчанию, чтобы сразу иметь возможность устанавливать пакеты из моего локального репозитория. Для этого я создал файл /ovz/postinstall/setup.d/01apt с таким содержимым:

```
cat >> /etc/apt/sources.list.d/sisyphus.local.list <<END

# Local Sisyphus

rpm [alt] ftp://192.168.46.1/distrib/linux/alt-linux-sisyphus i586 classic
rpm-src [alt] ftp://192.168.46.1/distrib/linux/alt-linux-sisyphus i586 classic
rpm [alt] ftp://192.168.46.1/distrib/linux/alt-linux-sisyphus noarch classic
rpm-src [alt] ftp://192.168.46.1/distrib/linux/alt-linux-sisyphus noarch classic

END
```

Затем я выполнил команду:

```
spt -v --noiso --image-type=tgz --maketty ~/ovz/
```

и получил файл /ovz/out/altlinux, который можно использовать как шаблон виртуального сервера для OpenVZ. Теперь файл /ovz/out/altlinux необходимо скопировать на ведущий узел кластера с именем /var/lib/vz/template/cache/altlinux-sisyphus.tar.gz и выполнить следующее:

```
[root@m1 ~]# vzctl create 101 --ostemplate altlinux-sisyphus --config vps.basic
Creating VE private area: /var/lib/vz/private/101
Performing postcreate actions
VE private area was created
[root@m1 ~]# vzctl set 101 --name router --save
Name router assigned
Saved parameters for VE 101
[root@m1 ~]# vzctl set router --onboot yes --save
Saved parameters for VE 101
[root@m1 ~]# vzctl set router --hostname router.mydomain.com --save
Set hostname: router.mydomain.com
Saved parameters for VE 101
```

Таким образом, мы создали виртуальный сервер, задали для него имя, указали, что он должен загружаться при старте OpenVZ, и присвоили ему FQDN. Везде мы использовали ключ -save, чтобы сохранить внесенные изменения после перезапуска виртуального сервера.

Далее необходимо пробросить физический интерфейс eth1, который мы оставили незадействованным на этапе конфигурирования узлов кластера, в виртуальный сервер, чтобы сделать его доступным извне:

```
[root@m1 ~]# vzctl set router --netdev_add eth1 --save
Saved parameters for VE 101
```

Теперь можно запустить виртуальный сервер, войти в него, сконфигурировать сетевой интерфейс eth1 и проверить доступность физических серверов из виртуального и наоборот:

```
[root@m1 ~]# vzctl start router
Starting VE ...
VE is mounted
Setting CPU units: 1000
Set hostname: router.mydomain.com
VE start in progress...
[root@m1 ~]# vzctl enter router
entered into VE 101
[root@router /]# ip address add 192.168.46.200/24 dev eth1
[root@router /]# ip link set eth1 up
[root@router /]# ping 192.168.46.1
PING 192.168.46.1 (192.168.46.1) 56(84) bytes of data.
64 bytes from 192.168.46.1: icmp_seq=1 ttl=64 time=5.37 ms

--- 192.168.46.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.376/5.376/5.376/0.000 ms
```

Команды внутри виртуального сервера можно также выполнять не переключаясь «вовнутрь» виртуального сервера (vzctl enter), с помощью vzctl exec:

```
[root@m1 ~]# vzctl exec router ping 192.168.46.1
PING 192.168.46.1 (192.168.46.1) 56(84) bytes of data.
64 bytes from 192.168.46.1: icmp_seq=1 ttl=64 time=6.34 ms
```

Для сохранения сетевых настроек после перезагрузки в ALT Linux используется система etcnet, о которой много написано на `http://etcnet.org/` и `http://wiki.sisyphus.ru/admin/etcnet`:

```
[root@router /]# mkdir /etc/net/ifaces/eth1
[root@router /]# echo 192.168.46.200/24 > /etc/net/ifaces/eth1/ipv4address
[root@router /]# echo default via 192.168.46.1 dev eth1 > /etc/net/ifaces/eth1/i
[root@router /]# echo "BOOTPROTO=static
> ONBOOT=yes
> TYPE=eth" > /etc/net/ifaces/eth1/options
[root@router /]# service network restart
Computing interface groups: ... 3 interfaces found
Processing /etc/net/vlantab: empty.
Stopping group 1/realphys (1 interfaces)
        Stopping eth1: ..OK
Stopping group 0/virtual (2 interfaces)
        Stopping lo: .OK
        Stopping venet0: .OK
error: setting key "net.ipv4.icmp_echo_ignore_broadcasts": Operation not permitt
error: setting key "net.ipv4.tcp_syncookies": Operation not permitted
error: setting key "net.ipv4.tcp_timestamps": Operation not permitted
Computing interface groups: ... 3 interfaces found
Starting group 0/virtual (2 interfaces)
        Starting lo: ....OK
        Starting venet0: ......OK
Starting group 1/realphys (1 interfaces)
        Starting eth1: ......OK
Processing /etc/net/vlantab: empty.
```

Сообщения «Operation not permitted» связаны с ограничениями для виртуального сервера, определенными по умолчанию, и в нашем случае на функционировании сети отрицательно не сказываются. Можно закомментировать соответствующие строки в файле /etc/net/sysctl.conf, чтобы эти сообщения больше не появлялись.

Созданный нами виртуальный сервер будет использовать сетевой интерфейс eth1 того узла, на котором он в данный момент работает, поэтому в случае отказа узла m1 сервер переместится на узел m2 и будет доступен там по тому же самому адресу. Можно смоделировать эту ситуацию и увидеть с внешнего физического сервера следующее:

```
$ ping 192.168.46.200
PING 192.168.46.200 (192.168.46.200) 56(84) bytes of data.
64 bytes from 192.168.46.200: icmp_seq=1 ttl=64 time=0.549 ms
...
From 192.168.46.1 icmp_seq=83 Destination Host Unreachable
```

```
...
64 bytes from 192.168.46.200: icmp_seq=179 ttl=64 time=1.05 ms

--- 192.168.46.200 ping statistics ---
179 packets transmitted, 25 received, +93 errors, 86% packet loss, time 178149ms
rtt min/avg/max/mdev = 0.298/193.970/1702.783/397.285 ms, pipe 3
```

6 Строим виртуальную сеть

В большинстве случаев на одном физическом сервере размещают несколько виртуальных серверов, при этом необходимо каким-то образом обеспечить доступность сервисов, работающих на виртуальных серверах, другим физическим машинам. Пробрасывать каждому виртуальному серверу по физическому интерфейсу накладно, да и неудобно. Часто хочется спрятать все виртуальные сервера за одним маршрутизатором/брандмауэром.

Посмотрим, какие средства для этого предлагает OpenVZ. Он предлагает 2 типа виртуальных сетевых интерфейсов:

- *venet* — соединения типа точка—точка между физическим сервером и виртуальным, которые создаются автоматически для каждого виртуального сервера и конфигурируются с помощью vzctl. Они не имеют MAC-адресов, не поддерживают широковещательные сообщения (broadcast), на них не работают снифферы и средства учета трафика, использующие libicap (например, tcpdump), на них нельзя строить бриджи.

- *veth* — соединения типа точка—точка между физическим сервером и виртуальным, которые нужно создавать и конфигурировать вручную средствами того дистрибутива Linux, который работает на физическом и виртуальном сервере. Они лишены ограничений venet (которые можно рассматривать как достоинства с точки зрения безопасности и эффективности) и выглядят как полноценные ethernet-интерфейсы.

Если в качестве маршрутизатора/брандмауэра для доступа к виртуальным серверам использовать физический сервер, стоит, несомненно, предпочесть venet. Поскольку такая конфигурация более распространена, то и venet-интерфейсы используются чаще. Однако чем сложнее конфигурация маршрутизатора/брандмауэра, тем больше оснований появляется для вынесения его в отдельный виртуальный сервер, чтобы не

перегружать физический сервер лишними задачами и лишним ПО. В нашем случае дополнительным поводом стало желание иметь один и тот же адрес виртуального сервера, доступный извне, независимо от адреса узла кластера, на котором он в данный момент работает. Эта конфигурация в некоторых случаях может оказаться еще более сложной, если, например, на проброшенном физическом интерфейсе организовать поддержку IEEE 802.1Q VLAN, чтобы принять несколько vlan-ов, но с точки зрения настройки OpenVZ эта конфигурация не будет ничем отличаться от того, что было рассмотрено выше — разница будет только в настройке проброшенного сетевого интерфейса. Более важным является то, что в случае использования в качестве маршрутизатора/брандмауэра виртуального сервера более удобным будет построить виртуальную сеть на veth-интерфейсах. Выглядеть это будет так, как показано на рисунке 3.

Рис. 3: Схема соединения виртуальных серверов

Итак, для каждого виртуального сервера мы создаем veth-интерфейс, а концы этих интерфейсов со стороны физического сервера объединяем в бридж — в результате получается аналог хаба, в который включены все виртуальные сервера. Один из них уже имеет проброшенный в него физический интерфейс — этот виртуальный сервер и будет играть роль маршрутизатора/брандмауэра.

Создаем и стартуем виртуальные сервера:

```
[root@m1 ~]# vzctl create 102 --ostemplate altlinux-sisyphus --config vps.basic
```

```
Creating VE private area: /var/lib/vz/private/102
Performing postcreate actions
VE private area was created
[root@m1 ~]# vzctl set 102 --name mail --save
Name ve1 assigned
Saved parameters for VE 102
[root@m1 ~]# vzctl set mail --onboot yes --save
Saved parameters for VE 102
[root@m1 ~]# vzctl set mail --hostname mail.mydomain.com --save
Saved parameters for VE 102
[root@m1 ~]# vzctl start mail
Starting VE ...
VE is mounted
Adding IP address(es): 192.168.199.2
Setting CPU units: 1000
Set hostname: mail.mydomain.com
VE start in progress...

[root@m1 ~]# vzctl create 103 --ostemplate altlinux-sisyphus --config vps.basic
Creating VE private area: /var/lib/vz/private/103
Performing postcreate actions
VE private area was created
[root@m1 ~]# vzctl set 103 --name dbms --save
Name ve2 assigned
Saved parameters for VE 103
[root@m1 ~]# vzctl set dbms --onboot yes --save
Saved parameters for VE 103
[root@m1 ~]# vzctl set dbms --hostname dbms.mydomain.com --save
Saved parameters for VE 103
[root@m1 ~]# vzctl start dbms
Starting VE ...
VE is mounted
Adding IP address(es): 192.168.199.3
Setting CPU units: 1000
Set hostname: dbms.mydomain.com
VE start in progress...
```

Создаем и конфигурируем veth-интерфейсы внутри виртуальных серверов:

```
[root@m1 ~]# vzctl set router --veth_add veth1,00:12:34:56:78:9A,eth0,00:12:34:5
```

```
Processing veth devices
Saved parameters for VE 101
[root@m1 ~]# vzctl exec router ip address add 192.168.199.1/24 dev eth0
[root@m1 ~]# vzctl exec router ip link set eth0 up

[root@m1 ~]# vzctl set mail --veth_add veth2,00:12:34:56:78:9C,eth0,00:12:34:56:
Processing veth devices
Saved parameters for VE 102
[root@m1 ~]# vzctl exec mail ip address add 192.168.199.2/24 dev eth0
[root@m1 ~]# vzctl exec mail ip link set eth0 up

[root@m1 ~]# vzctl set dbms --veth_add veth3,00:12:34:56:78:9E,eth0,00:12:34:56:
Processing veth devices
Saved parameters for VE 103
[root@m1 ~]# vzctl exec dbms ip address add 192.168.199.3/24 dev eth0
[root@m1 ~]# vzctl exec dbms ip link set eth0 up
```

Имена интерфейсов и их MAC-адреса мы придумываем сами, поэтому необходимо, чтобы последние не пересекались с существующими.

Объединяем концы veth-интерфейсов со стороны физического сервера в бридж:

```
[root@m1 ~]# ip link set veth1 up
[root@m1 ~]# ip link set veth2 up
[root@m1 ~]# ip link set veth3 up
[root@m1 ~]# brctl addbr br0
[root@m1 ~]# brctl addif br0 veth1
[root@m1 ~]# brctl addif br0 veth2
[root@m1 ~]# brctl addif br0 veth3
[root@m1 ~]# ip link set br0 up
```

Проверяем работоспособность внутренней виртуальной сети:

```
[root@m1 ~]# vzctl exec router ping 192.168.199.2
PING 192.168.199.2 (192.168.199.2) 56(84) bytes of data.
64 bytes from 192.168.199.2: icmp_seq=1 ttl=64 time=15.9 ms

[root@m1 ~]# vzctl exec router ping 192.168.199.3
PING 192.168.199.3 (192.168.199.3) 56(84) bytes of data.
64 bytes from 192.168.199.3: icmp_seq=1 ttl=64 time=3.71 ms
```

Теперь на виртуальных серверах описываем маршрут во внешнюю физическую сеть:

```
[root@m1 ~]# vzctl exec mail ip route add 192.168.0.0/16 via 192.168.199.1
[root@m1 ~]# vzctl exec dbms ip route add 192.168.0.0/16 via 192.168.199.1
```

На виртуальном маршрутизаторе включаем пересылку пакетов между физической и виртуальной сетями:

```
[root@m1 ~]# vzctl exec router sysctl -w net.ipv4.ip_forward=1
```

Теперь если на машине из физической сети 192.168.46.0/24 описать маршрут в сеть 192.168.199.0/24 (или настроить NAT на виртуальном маршрутизаторе), мы получим то, чего и добивались:

```
[root@m1 ~]# vzctl exec mail ping 192.168.46.1
PING 192.168.46.1 (192.168.46.1) 56(84) bytes of data.
64 bytes from 192.168.46.1: icmp_seq=1 ttl=63 time=0.982 ms
```

Желательно, чтобы все описанные выше настройки сохранялись при перезапуске виртуальных серверов, сервиса vz и ведущего узла кластера. С настройками виртуальных серверов проще всего — их можно сохранить в etcnet:

```
[root@m1 ~]# vzctl enter mail
[root@mail /]# mkdir /etc/net/ifaces/eth0
[root@mail /]# echo 192.168.199.2/24 > /etc/net/ifaces/eth0/ipv4address
[root@mail /]# echo 192.168.0.0/16 via 192.168.199.1 dev eth0 > /etc/net/ifaces/
[root@mail /]# echo "BOOTPROTO=static
> ONBOOT=yes
> TYPE=eth" > /etc/net/ifaces/eth0/options

[root@m1 ~]# vzctl enter dbms
[root@dbms /]# mkdir /etc/net/ifaces/eth0
[root@dbms /]# echo 192.168.199.3/24 > /etc/net/ifaces/eth0/ipv4address
[root@dbms /]# echo 192.168.0.0/16 via 192.168.199.1 dev eth0 > /etc/net/ifaces/
[root@dbms /]# echo "BOOTPROTO=static
> ONBOOT=yes
> TYPE=eth" > /etc/net/ifaces/eth0/options
```

Для автоматического добавления конца veth-интерфейсов со стороны физического сервера в бридж при старте соответствующего виртуального сервера потребуется исправить скрипт /usr/sbin/vznetcfg, добавив в конец функции `init_veth()` строку (сделать это нужно на двух узлах кластера):

```
brctl addif br0 ${dev}
```

В будущем разработчики OpenVZ обещают доработать этот скрипт, чтобы подобного рода измения можно было описывать в конфигурационном файле.

Наконец, бридж тоже нужно создать, и сделать это необходимо еще до старта всех виртуальных серверов. Лучше всего добавить его создание в конфигурацию etcnet на обеих узлах кластера:

```
[root@m1 ~]# mkdir /etc/net/ifaces/br0
[root@m1 ~]# echo TYPE=bri > /etc/net/ifaces/br0/options
```

Есть одна неприятная деталь. В конфигурацию виртуальных серверов мы добавили маршрут в сеть 192.168.0.0/16, но во многих случаях нам потребуется добавить туда маршрут по уполчанию. Сделать этого мы не сможем, так как такой маршрут, созданный OpenVZ заранее для собственных нужд, уже есть:

```
[root@m1 ~]# vzctl exec mail ip route
192.168.199.0/24 dev eth0  proto kernel  scope link  src 192.168.199.2
192.0.2.0/24 dev venet0  scope host
192.168.0.0/16 via 192.168.199.1 dev eth0
default via 192.0.2.1 dev venet0
```

Он создается и автоматически привязывается к интерфейсу venet0. Ни эта довольно навязчивая автоматика, ни venet-интерфейсы вообще нам сейчас не нужны, мы используем только veth. Поэтому чтобы такого не происходило, потребуется исправить скрипт, выполняющий настройку сетевых интерфейсов виртуального сервера. В случае ALT Linux Sisyphus это `/etc/vz/dists/scripts/etcnet-add_ip.sh`. В нем нам нужно модифицировать функцию `add_ip()` таким образом, чтобы она выполнялась только при наличии присвоенного venet-интерфейсу адреса:

```
add_ip()
{
    local i ip

    if [ -n "$IP_ADDR" ]; then

        if [ "$VE_STATE" = "starting" ]; then
            setup_network
        fi
```

```
        backup_configs "$IPDELALL"

        i=0
        for ip in ${IP_ADDR}; do
            i="$(find_unused_alias "$((i+1))")"
            create_alias "$ip" "$i"
        done

        move_configs

        if [ "$VE_STATE" = "running" ]; then
            # synchronyze config files & interfaces
            ifdown "$VENET_DEV"
            ifup "$VENET_DEV"
        fi

    fi
}
```

Затем в виртуальных серверах необходимо удалить из etcnet настройки интерфейса venet0:

```
[root@m1 ~]# vzctl exec router rm -rf /etc/net/ifaces/venet0
[root@m1 ~]# vzctl exec mail rm -rf /etc/net/ifaces/venet0
[root@m1 ~]# vzctl exec dbms rm -rf /etc/net/ifaces/venet0
```

Теперь можно перезапустить сервис vz — в конфигурации виртуальных серверов останутся только те маршруты, которые мы указали явно.

Таким образом мы добились того, чего хотели: в штатном режиме виртуальные сервера mail и dbms работают на узле m1, а в случае его отказа автоматически переезжают на узел m2, при этом с точки зрения внешнего наблюдателя из физической сети 192.168.46.0/24 наблюдается лишь кратковременный перерыв в обслуживании:

```
$ ping 192.168.199.2
PING 192.168.199.2 (192.168.199.2) 56(84) bytes of data.
64 bytes from 192.168.199.2: icmp_seq=1 ttl=64 time=0.549 ms
...
From 192.168.46.1 icmp_seq=83 Destination Host Unreachable
...
```

```
From 192.168.46.200 icmp_seq=83 Destination Host Unreachable
...
64 bytes from 192.168.199.2: icmp_seq=179 ttl=64 time=1.05 ms

--- 192.168.46.200 ping statistics ---
179 packets transmitted, 25 received, +93 errors, 86% packet loss, time 178149ms
rtt min/avg/max/mdev = 0.298/193.970/1702.783/397.285 ms, pipe 3
```

7 Что дальше

Итак, мы выполнили базовую настройку двухузлового кластера, подняли систему виртуализации OpenVZ, кластеризовали ее, а затем настроили несколько виртуальных серверов, виртуальную сеть между ними, и связали ее с внешней физической сетью. При этом мы покзаали преимущества такого способа построения систем как с точки зрения надежности, так и с точки зрения снижения затрат на оборудование и его обслуживание. Теперь можно углубляться в детали: садиться и вдумчиво читать OpenVZ Users Guide — `http://download.openvz.org/doc/OpenVZ-Users-Guide.pdf`[1].

Для полноценного использования OpenVZ нам потребуется настроить:

- квоты на процессорное время для виртуальных серверов;
- квоты потребление системных ресурсов виртуальными серверами (количество процессов, количество сокетов, объем виртуальной памяти, различных буферов и т. д.);
- дисковые квоты;
- доступ к физическим устройствам и файловым системам физического сервера, если в этом есть необходимость.

В отличие от основного материала статьи, все перечисленное очень подробно описано в документации, а просто пересказывать документацию я не стану.

И наконец, после более тщательной настройки OpenVZ можно переходить к настройке самих виртуальных серверов (либо к миграции существующих физических в OpenVZ) — но те, кто все-таки дочитал статью до конца, думаю, сумеют это сделать.

[1]Кстати, перевод этого руководства доступен на `http://www.opennet.ru/docs/RUS/virtuozzo/`, однако он довольно плохого качества — лучше все-таки читать оригинал

8 Ссылки и источники

- http://linux-ha.org
- http://drbd.org
- http://openvz.org
- http://altlinux.ru
- http://sisyphus.ru
- http://freesource.info