Кластеризация + виртуализация : HAD + DRBD + 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 решение должно быть:

- целиком построено из свободного программного обеспечения
- максимально простым, надежным и не требовать слишком больших затрат на «железо»

Первое требование автоматически исключает из рассмотрения проприетарные средства кластеризации, второе также ставит под сомнение осмысленность применения таких комплексных решений как Linux-HA и Red Hat Cluster Manager, которые являются слишком сложными. Более разумным представляется сочетание простого менеджера кластера,

работающего в режиме active/passive (один узел работает, другой простаивает, ожидая отказа первого), и аналогичного active/passive механизма организации файлового хранилища кластера. Такое решение можно построить с помощью менеджера кластера HAD (это основной компонент продукта JET Infosystems High Availabilty Cluster, являющегося дальнейшим развитием FreeHA — `http://http://bolthole.com/freeha/` — а его развитием занимается член ALT Linux Team и сотрудник JET Infosystems Сергей Пинаев) и DRBD (средство автоматического зеркалирования разделов жесткого диска на двух компьютерах — `http://drbd.org`).

Что же касается виртуализации, то здесь выбор очень широк, 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 DRBD и Heartbeat (менеджером кластера проекта Linux HA) — `http://wiki.openvz.org/HA_cluster_with_DRBD_and_Heartbeat`.

Наконец, главное: HAD, DRBD и OpenVZ работают в ALT Linux «из коробки».

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 HAD и DRBD. 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 не зависит друг от друга.

4. *Кластеризация + виртуализация* — общий физический сервер (точнее, один единственный сервис — система виртуализации) дублируется средствами HAD и DRBD, 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.

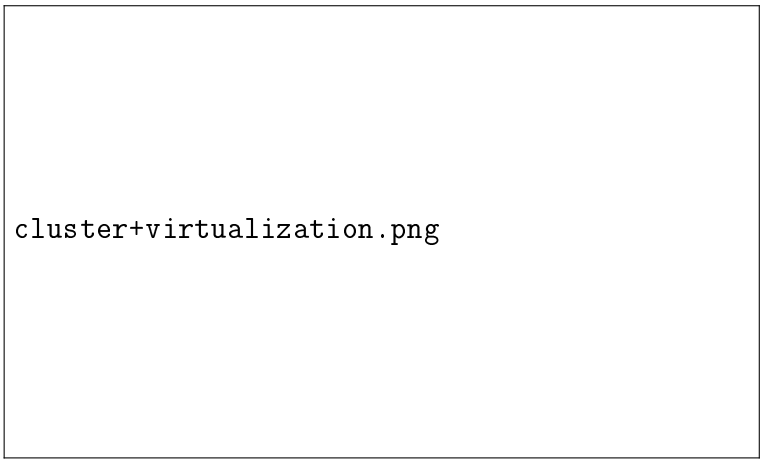

Рис. 1: Кластеризация + виртуализация

3 Строим кластер

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

карты:

Рис. 2: Схема кластера

- `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-wks-smp - Linux drbd kernel modules for DRBD.
kernel-source-drbd-0.7.22 - Kernel source for DRBD
```

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  2.6.18-std-smp-alt7 #1 SMP Sat Aug 4 01:05:35 MSD 2007 x8
```

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

```
[root@m1 ~]# apt-get install kernel-modules-drbd-std-smp 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 C;
```

```
    # параметры синхронизации
    syncer {
# скорость синхронизации (Mbyte/sec)
rate  125M;                 # большего из GigaBit Ethernet не выжать
    }

    # описание узла m1
    on m1.mydomain.com {
        device     /dev/drbd0;          # имя drbd-устройства
        disk       /dev/sda3;           # 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/sda3;            # 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.

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: 0.7.22 (api:79/proto:74)
SVN Revision: 2554 build by builder@hint1.office.altlinux.org, 2007-08-04 01:47:
 0: cs:Connected st:Secondary/Secondary ld:Consistent
     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/sda3, 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: 0.7.22 (api:79/proto:74)
SVN Revision: 2554 build by builder@hint1.office.altlinux.org, 2007-08-04 01:47:
 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: 0.7.22 (api:79/proto:74)
SVN Revision: 2554 build by builder@hint1.office.altlinux.org, 2007-08-04 01:47:
 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/sda3 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 secondary all
[root@m2 ~]# drbdadm primary all
[root@m2 ~]# mount /dev/drbd0 /d0
```

В случае отказа узла m1 достаточно выполнить только те команды, которые приведены выше для узла m2.

3.2 Автоматическое отслеживание состояния узлов: HAD

На каждом узле кластера устанавливаем HAD штатным для ALT Linux способом:

```
[root@m1 ~]# apt-get install had
```

Создаем конфигурационный файл HAD (/etc/had/had.conf), который на узле m1 будет выглядеть так:

```
ENABLE=yes
DEBUG=no
IP1="192.168.200.1"
NET1="192.168.200.255"
NODES=2
MONITORING=1
TIMEOUT=5
```

а на m2 — так:

```
ENABLE=yes
DEBUG=no
IP1="192.168.200.2"
NET1="192.168.200.255"
NODES=2
MONITORING=1
TIMEOUT=5
```

Параметры в конфигурационом файле — это переменные shell, используемые стартовым скриптом /etc/init.d/had, который передает их в виде параметров командной строки исполняемому файлу /usr/sbin/had.

HAD отслеживает следующие основные состояния узла кластера:

1. *RUNNING* — активный узел (всегда один)
2. *STANDBY* — корректно функционирующий пассивный узел
3. *ERRORED* — некорректно функционирующий узел
4. *TIMEDOUT* — узел, связь с которым потеряна

Соответственно, в процессе работы HAD вызывает 3 скрипта:

1. */etc/had/hasrv/starthasrv* — при переключении в состояние 1
2. */etc/had/hasrv/stophasrv* — при переключении в состояние 3 или при завершении работы HAD
3. */etc/had/hasrv/monitorhasrv* — для проверки корректности функционирования кластера

В штатном режиме кластер стартует только в том случае, если достигнут кворум, т.е. стартовали более половины узлов (для двух узлов это 2/2+1=2, для четырех — 4/2+1=3 и т.д. — впрочем, сейчас нам интересны только двухузловые конфигурации). Чтобы запустить кластер, состоящий из одного узла (это может потребоваться разве что при выходе второго узла из строя и необходимости перезагрузки первого), придется выполнить:

```
[root@m1 ~]# service had start-force-main
```

или:

```
[root@m1 ~]# /usr/sbin/had -m ...
```

После старта кластера имена узлов сортируются в алфавитном порядке, на каждом вызывается скрипт 3 и выполняются следующие действия:

- если все узлы неактивны (вернули не 0), первый по алфавиту переводится в состояние 1, остальные узлы переводятся в состояние 2
- если активны более одного узла (вернули 0), только первый по алфавиту переводится в состояние 1, а остальные активные — в состояние 3

Эти же действия выполняются далее в процессе работы кластера: например, отказал один из узлов (тогда первый по алфавиту из оставшихся узлов в состоянии 2 перейдет в состояние 1) или была потеряна связь с узлом (и он, оставшись в одиночестве, перешел в состояние 1 — при возвращении в кластер он или другой узел будет переведен в состояние 3). Более подробно о состояниях и переходах между ними можно узнать из документации в пакете had.

В нашем случае скрипты, вызываемые при переходе из одного состояния в другое, будут выглядеть так:

```
[root@m1 ~]# cat /etc/had/hasrv/starthasrv
#!/bin/sh -e

drbdadm primary all
mount /dev/drbd0 /d0

[root@m1 ~]# cat /etc/had/hasrv/stophasrv
#!/bin/sh

umount /dev/drbd0
drbdadm secondary all

[root@m1 ~]# cat /etc/had/hasrv/monitorhasrv
#!/bin/sh

[ `drbdadm state all | awk -F'/' '{print $1}'` == "Primary" ] && \
grep drbd0 /proc/mounts > /dev/null && \
exit 0
exit 1
```

Теперь можно запустить HAD на каждом узле:

```
[root@m1 ~]# service had start
```

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

```
m1 JETHA[31423]: daemon starting
m1 JETHA[31423]: Waiting for quorum of nodes to come online
m1 had: had startup succeeded
m1 JETHA[31423]: Services not running, cleaning up
```

```
m1 JETHA[31423]: Stopping Services (stophasrv)
m1 JETHA[31423]: Startup done, starting main loop [5]
m1 JETHA[31423]: Starting Services (starthasrv)
m1 kernel: drbd0: Secondary/Secondary --> Primary/Secondary
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 JETHA[31423]: ok, we are RUNNING!
```

В логах ведомого узла можно будет увидеть следующее:

```
m2 JETHA[27544]: daemon starting
m2 JETHA[27544]: Waiting for quorum of nodes to come online
m2 had: had startup succeeded
m2 JETHA[27544]: Services not running, cleaning up
m2 JETHA[27544]: Stopping Services (stophasrv)
m2 JETHA[27544]: Startup done, starting main loop [5]
m2 kernel: drbd0: Secondary/Secondary --> Secondary/Primary
```

После старта HAD на обоих узлах кластера устройство /dev/drbd0 будет примонтировано на ведущем узле. Если остановить сервис had на ведущем узле m1 (или выключить узел m1), ведомый m2 возьмет на себя функции ведущего (перейдет в состояние 1), и устройство /dev/drbd0 будет примонтировано на нем, при этом в логах мы увидим:

```
m2 kernel: drbd0: Secondary/Primary --> Secondary/Secondary
m2 JETHA[27544]: Warning: Node m1.mydomain.com timeout
m2 JETHA[27544]: Starting Services (starthasrv)
m2 kernel: drbd0: Secondary/Secondary --> Primary/Secondary
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 JETHA[27544]: ok, we are RUNNING!
```

При повторном запуске сервиса had на m1 или при включении узла m1 он не станет ведущим, а перейдет в состояние 2, ведущим останется узел m2.

Текущее состояние узлов можно узнать так:

```
[root@m1 ~]# service had status
HAD running. Node status:
m1.mydomain.com [Standby]
m2.mydomain.com [Running]
```

4 Строим систему виртуализации и кластеризуем ее

Скрипты, вызываемые при переходе узлов кластера из одного состояния в другое, могут быть дополнены списком сервисов, для которых необходимо гарантировать отказоустойчивость, а конфигурационные файлы и файлы данных этих сервисов могут быть перемещены на файловую систему, созданную поверх drbd-устройства. Можно также сконфигурировать алиас для сетевого интерфейса, который автоматически создается на ведущем узле — тем самым все сервисы будут 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 использования различных сервисов на одном сервере я и не говорю — это специфично не только для кластерных конфигураций.

Самым простым и изящным выходом в данной ситуации является виртуализация. Если использовать этот подход, нам потребуется обеспечить отказоустойчивость только для одного сервиса — системы виртуализации, в которой 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 ядро с поддержкой OpenVZ собрано отдельно, поэтому, если оно еще не установлено, его необходимо установить в дополнение к существующему ядру или вместо него, 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/sda2
    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/
-rw-r--r-- 1 root root  993322 Aug  4 01:05 System.map-2.6.18-ovz-smp-alt15
-rw-r--r-- 1 root root     512 Aug 14 22:33 boot.0800
-rw-r--r-- 1 root root   63357 Aug  4 01:00 config-2.6.18-ovz-smp-alt15
-rw------- 1 root root  421499 Aug 17 10:57 initrd-2.6.18-ovz-smp-alt15.img
lrwxrwxrwx 1 root root      31 Aug 17 10:57 initrd-smp.img -> initrd-2.6.18-ovz-
lrwxrwxrwx 1 root root      31 Aug 17 10:57 initrd.img -> initrd-2.6.18-ovz-smp-
-rw------- 1 root root   36352 Aug 17 10:59 map
lrwxrwxrwx 1 root root      28 Aug 17 10:57 vmlinuz -> vmlinuz-2.6.18-ovz-smp-al
-rw-r--r-- 1 root root 1615274 Aug  4 01:05 vmlinuz-2.6.18-ovz-smp-alt15
lrwxrwxrwx 1 root root      28 Aug 17 10:57 vmlinuz-smp -> vmlinuz-2.6.18-ovz-sm
```

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

После остановки HAD 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 отредактировать скрипты, вызываемые при переходе узлов кластера из одного состояния в другое, так:

```
[root@m1 ~]# cat /etc/had/hasrv/starthasrv
#!/bin/sh -e

drbdadm primary all
mount /dev/drbd0 /d0
/etc/init.d/vz start

[root@m1 ~]# cat /etc/had/hasrv/stophasrv
#!/bin/sh

/etc/init.d/vz stop
umount /dev/drbd0
drbdadm secondary all
```

После перезапуска HAD 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.

Кроме того, поскольку для шаблон - это запакованный в tar.gz архив корневой системы Linux, его нетрудно изготовить самостоятельно. В ALT Linux штатным способом изготовления такого архива является spt.

Шаблоны виртуальных серверов должны находиться в каталоге /var/lib/vz/template/cach
Если требуется использовать какой-либо шаблон по умолчанию (т.е. не

указывать его имя при создании виртуального сервера), его необходимо вписать в параметр DEF_OSTEMPLATE в файле /etc/vz/vz.conf.

Создаем первый виртуальный сервер:

```
[root@m1 ~]# vzctl create 101
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.

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
```

Рис. 3: Схема соединения виртуальных серверов

```
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
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 --netif_add eth0 --save
Configure meminfo: 35000
Configure veth devices: veth101.0
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 --netif_add eth0 --save
Configure meminfo: 35000
Configure veth devices: veth102.0
Saved parameters for VE 101
[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 --netif_add eth0 --save
Configure meminfo: 35000
Configure veth devices: veth103.0
Saved parameters for VE 101
[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
```

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

```
[root@m1 ~]# ip link set veth101.0 up
[root@m1 ~]# ip link set veth102.0 up
[root@m1 ~]# ip link set veth103.0 up
[root@m1 ~]# brctl addbr br0
[root@m1 ~]# brctl addif br0 veth101.0
[root@m1 ~]# brctl addif br0 veth102.0
[root@m1 ~]# brctl addif br0 veth103.0
[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 default via 192.168.199.1
[root@m1 ~]# vzctl exec dbms ip route add default 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 — для этого потребуется в HN в файле /etc/vz/vz.conf в параметре IPTABLES указать необходимые модули), 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 router
[root@router /]# mkdir /etc/net/ifaces/eth0
[root@router /]# echo 192.168.199.1/24 > /etc/net/ifaces/eth0/ipv4address
[root@router /]# echo "BOOTPROTO=static
> ONBOOT=yes
> TYPE=eth" > /etc/net/ifaces/eth0/options

[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 default via 192.168.199.1 dev eth0 > /etc/net/ifaces/eth0/ip
[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 default via 192.168.199.1 dev eth0 > /etc/net/ifaces/eth0/ip
[root@dbms /]# echo "BOOTPROTO=static
> ONBOOT=yes
> TYPE=eth" > /etc/net/ifaces/eth0/options
```

Для автоматического добавления концов veth-интерфейсов со стороны физического сервера в бридж при старте соответствующего виртуального сервера потребуется создать файл /etc/vz/vznet.conf со следующим содержимым:

```
EXTERNAL_SCRIPT=/etc/vz/vznet.sh
```

Внешний скрипт для конфигурирования veth-интерфейсов /etc/vz/vznet.sh будет выглядеть так:

```
#!/bin/sh

brctl addif br0 $3
ip link set $3 up
```

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
```

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 показали 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;

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

- 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.

8 Ссылки и источники

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