HA-кластеры

Рассмотрим создание высокодоступных кластеров на основе pacemaker.

Оглавление

[Высокодоступный кластер (HA cluster)](#_3dy6vkm)

[High-availability (высокодоступные) кластеры](#_wneujyd1z21c)

[Компоненты, необходимые для HA-кластера](#_w50za8i2lgfz)

[Общее хранилище](#_ep1cnlwxfys3)

[Разделение сетей](#_5gvef6zerxil)

[Сетевой бондинг/тиминг](#_z078ifyj8i0u)

[Multipathing](#_hbdbxa0kgiz)

[Fencing/STONITH устройства и кворум](#_va5ner6r9a0d)

[Pacemaker](#_yao3lw306df3)

[Resource Agents](#_6219wo87yd1f)

[Corosync](#_9jgj28vkhtjp)

[Базовая настройка кластера](#_3t7bxgkbd4vp)

[Кворум](#_xrqqrdnxyzro)

[Практическое задание](#_4i7ojhp)

[Дополнительные материалы](#_2xcytpi)

[Используемая литература](#_1ci93xb)

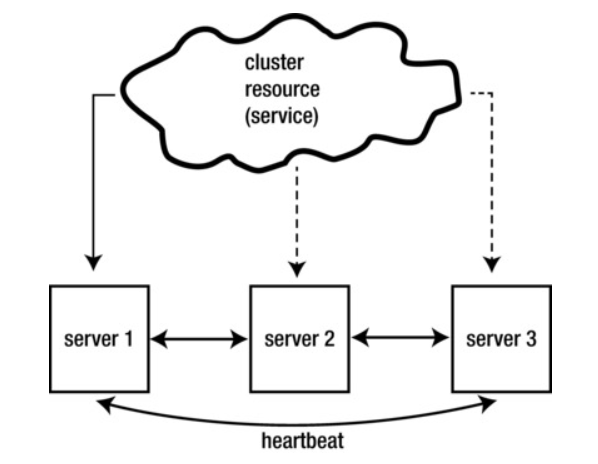
# Высокодоступный кластер (HA cluster)

Кластер — набор серверов, работающих вместе над одной задачей или оказывающих один и тот же сервис. То, какой это сервис, какая задача выполняется и каким образом, зависит от самого кластера. Их можно разделить на три вида.

## High-availability (высокодоступные) кластеры

Основная цель такого рода кластеров — продолжать обслуживать пользователей без перерыва, если происходят какие-либо проблемы с одним или более компонентом кластера — то есть предлагать высокодоступный сервис. Участники таких кластеров должны проверять состояние соседей по кластеру и предпринимать соответствующие меры (например миграция сервиса, если потребуется) в случае, когда какая-либо нода кластера становится нездоровой и не может обслуживать запросы. HA-кластеры можно еще разделить на два подвида:

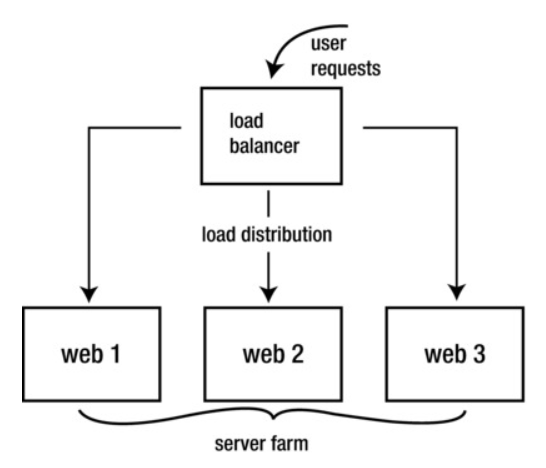
* Active-Active HA — сервис запущен на всех нодах кластера и любая из нод может обслуживать запросы клиентов.
* Active-Standby HA — сервис работает только на одной ноде — активной, вторая (и последующие, если есть) находится в горячем резерве и будет использована, если текущая активная нода будет не в состоянии обслуживать клиентов.



Частный случай HA-кластера — storage-кластер. Такие кластеры предоставляют сервисы хранения данных. При этом все члены кластера предлагают пользователю единую распределенную высокодоступную файловую систему. Примером может быть кластер с GFS2-файловой системой.

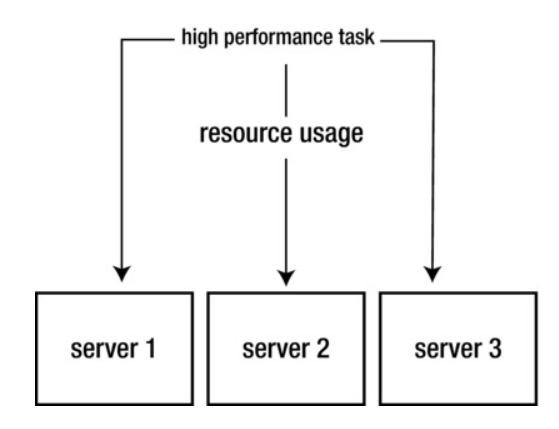
Load-balancing-кластеры. Необходимы для распределения нагрузки между серверами. Такие балансировщики нагрузки могут быть как Active-Active, так и Active-Standby. Кластеры балансировки нагрузки обычно используются в средах с высокой нагрузкой,например, для очень популярных веб-сайтов. Цель кластера балансировки нагрузки — перераспределение задачи на сервер, у которого есть ресурсы для её выполнения. Это похоже на высокопроизводительную кластеризацию, но разница в том, что в высокопроизводительных кластерах, как правило, все серверы работают над одной и той же задачей, а кластеры с балансировкой нагрузки заботятся о её распределении, чтобы получить оптимальную эффективность обработки задач.

Кластер балансировки нагрузки состоит из двух объектов: балансировщика нагрузки и фермы серверов, лежащих за ним. Балансировщик нагрузки получает запросы от конечных пользователей и перераспределяет их на один из серверов, доступных в ферме. В Linux это может быть Linux Virtual Server (LVS) или, например, HAProxy. Балансировщики нагрузки также отслеживают доступность серверов в ферме серверов, чтобы определить, где можно разместить ресурсы.



High-performance-кластеры. Кластеры, необходимые для высокой производительности. Все участники кластера выглядят как одно устройство и необходимы, когда требуется большая вычислительная мощность, например, в заданиях на рендеринг или сложных научных вычислениях, слишком ресурсоемких, чтобы их можно было обрабатывать одним сервером. В такой ситуации работа может выполняться несколькими серверами, чтобы обеспечить её плавное и своевременное выполнение.

Подход к высокопроизводительной кластеризации — использование образа единой системы (Single System Image — SSI). Используя этот подход, кластер рассматривает несколько компьютеров как один, и выделяет ресурсы и запрашивает их там, где они доступны. Высокопроизводительная кластеризация используется в определенных средах и не так широко распространена, как кластеризация высокой доступности.



# Компоненты, необходимые для HA-кластера

Чтобы построить кластер высокой доступности, вам понадобится не только несколько серверов, которые связаны между собой. Как правило, в большинстве кластеров используются следующие компоненты:

1. Общее хранилище (Shared Storage).
2. Разные сети.
3. Бондинг/тиминг сетевых адаптеров.
4. Multipath для сети хранения данных.
5. Fencing/STONITH-устройства.

В зависимости от того, какой сервис представляет кластер, вам следует решить, какие компоненты необходимы для создания оптимального решения.

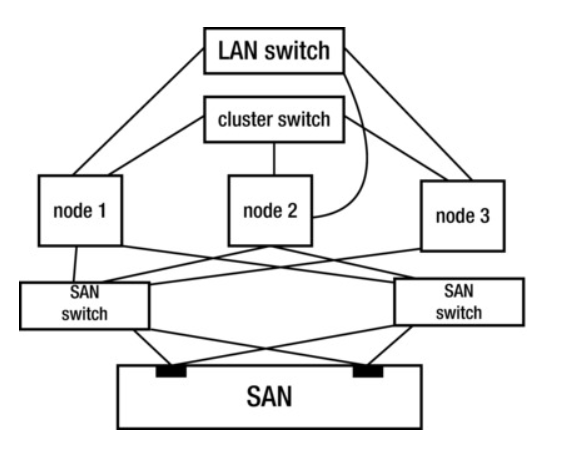
## Общее хранилище

Именно кластер определяет, на каком сервере будут размещаться общие для кластера ресурсы. Данные, размещённые на этом сервере, должны быть доступны остальным участникам кластера, даже если текущий сервер будет отключен. Вот почему большинству кластеров требуется общее хранилище. Однако бывают и исключения. Некоторые сервисы на самом деле не имеют большого количества файлов, которые должны быть синхронизированы с другими нодами кластера. Если ваш сервис работает только со статическими файлами, вы можете скопировать эти файлы вручную или настроить автоматическую синхронизацию файлов. Но большинству HA-кластеров проще использовать общее хранилище данных.

Грубо говоря, есть два подхода к работе с общим хранилищем. Вы можете использовать сетевую файловую систему (NFS) или сеть хранения данных (SAN). В NFS один или несколько каталогов общедоступны при помощи сети. Это самый простой способ настройки общего хранилища, но он не обеспечивает максимальной гибкости. Вот почему многие кластеры настроены с SAN (FIbre-Channel или iSCSI).

## Разделение сетей

Вы можете создать кластер, и весь трафик будет проходить по одной сети. Однако это не очень эффективно, потому как какой-либо клиент, используя слишком интенсивную передачу данных, может перегрузить сетевой интерфейс, что повлияет на служебный трафик кластера. Чтобы избежать такой ситуации, сеть, используемая для служебного трафика самого кластера, зачастую физически отделена от сети, которая используется для обслуживания клиентов. То есть типичный кластер имеет несколько сетевых подключений.



Существует пользовательская сеть, из которой внешние пользователи получают доступ к ресурсам кластера. Затем — выделенная сеть для пакетов протокола кластера. Она должна обеспечивать наилучшую возможную избыточность и гарантировать, что кластерный трафик не будет отброшен.

В-третьих, как правило, существует сеть хранения данных. То, как настроена эта сеть хранения, зависит от типа хранилища, которое вы используете. В сети Fibre Channel SAN узлы кластера будут иметь Host Bus Adapter (HBA) для подключения к сети Fibre Channel SAN. В сети iSCSI трафик SAN проходит по сети Ethernet, и для сети хранения не требуется ничего особенного, кроме выделенной инфраструктуры.

## Сетевой бондинг/тиминг

Чтобы подключить узлы кластера к сети, вы, конечно, можете использовать только один сетевой интерфейс. Если он выйдет из строя, узел потеряет соединение и возможность обслуживать пользователей, и кластер на это отреагирует. Поскольку кластер всё же связан с высокой доступностью, это обычно не то, чего вы хотите достичь.

Решение — в использовании сетевого бонда. Это гарантирует, что если один интерфейс выйдет из строя, другой — продолжит обслуживать пользователей.

## Multipathing

Когда нода кластера подключена к SAN, обычно существует несколько физических путей, по которым нода может видеть LUN в SAN. Это приводит к тому, что нода видит несколько устройств, хотя по факту это один и тот же LUN.

В конфигурации, где нода подключена к двум разным коммутаторам SAN, которые, в свою очередь, подключены к двум разным контроллерам SAN, будет четыре разных пути. В результате нода увидит не один диск, а четыре. Но использовать их все одновременно не получится, так как в один момент времени блочное устройство может контролироваться только одним контроллером. Драйвер Multipath обнаружит, что четыре разных диска фактически являются одним и тем же, и создаст абстрактное устройство, с которым уже будет взаимодействовать операционная система. Как правило, это устройство будет иметь имя вида mpatha. В результате можно будет не только гарантировать доступность диска в случае с проблемами в SAN-сети, но и балансировать нагрузку, так как multipath-демон может отправлять разные I/O по разным путям.

## Fencing/STONITH устройства и кворум

В кластере необходимо избегать ситуации, называемой split-brain. Это означает, что кластер разделен на две (или более) части, но обе части думают, что они — единственная оставшаяся в живых часть кластера. Чаще всего это приводит к ситуации, когда кластер становится недоступным, либо к повреждению файловой системы, если обе части кластера питаются записать данные в общий диск. Поскольку цель кластера высокой доступности — недопущение ситуаций, когда данные могут быть потеряны, это должно быть предотвращено, несмотря ни на что.

Чтобы предложить решение для сплит-брейна, есть два основных подхода. Во-первых, есть кворум. Кворум означает «большинство», и идею, лежащую в основе кворума, легко понять: если у кластера или его части нет кворума, то эта часть не будет предпринимать никаких действий. Этого решения не всегда достаточно, особенно в ситуации, когда обе части могут начать писать в один и тот же диск одновременно, также используется другой механизм. Этот механизм известен как STONITH (что означает shoot the other node in the head) или fencing (огораживание). Оба термина, STONITH и fencing, относятся к одному и тому же решению.

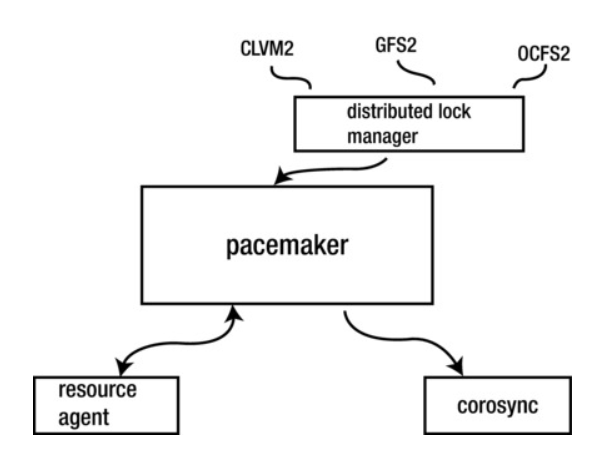
В STONITH определённое оборудование используется для отключения ноды, которая больше не реагирует на проверки кластера. Идея STONITH заключается в том, что перед миграцией ресурсов на другой узел в кластере, он должен подтвердить, что рассматриваемый узел действительно не работает. Для этого кластер отправит сигнал отключения на устройство STONITH, которое, в свою очередь, завершит работу не реагирующего узла. Это может казаться слишком радикальным подходом, но, так как он гарантирует, что повреждение данных никогда не произойдет, и может устранить некоторые временные ошибки (например, kernel crash), это не так уж плохо.

При настройке кластера вы должны решить, какой тип устройства STONITH вы хотите использовать. Это необходимо, поскольку STONITH обязателен в кластерах высокой доступности Linux. Доступны следующие различные типы устройств STONITH:

1. Интегрированные с менеджментом сервера (BMC), например, iLO, DRAC, IPMI.
2. Умные блоки распределения питания.
3. STONITH, основанный на внешнем общем диске.
4. STONITH для виртуальных машин, взаимодействующий с гипервизором.

# Pacemaker

Pacemaker — менеджер ресурсов кластера. Поэтому при создании кластера важно знать, каким образом Pacemaker взаимодействует с другими компонентами.



# Resource Agents

Для управления ресурсами используются агенты ресурсов (Resource Agents). Resource Agents — сценарий, который кластер использует для запуска, остановки и мониторинга ресурсов. Его можно сравнить с systemctl или runlevel-скриптом, но он был адаптирован для использования в кластере.

# Corosync

Corosync — фреймворк, который используется для обеспечения коммуникации между нодами кластера. Поддерживаемое количество нод в одном кластере — не более 16. Кроме этого, существует особый случай — когда количество нод в кластере равно двум.

Corosync требует особой конфигурации сети для поддержки связности между нодами кластера. Такой обмен сообщениями может быть реализован разными методами: стандартный unicast и multicast + IGMP, если требуется. Так как кластер представляет сети ваш ресурс в виде виртуального IP (VIP), сетевое оборудование должно поддерживать gratuitous ARP.

# Базовая настройка кластера

Для базовой настройки кластера нам следует установить несколько пакетов на 3 ноды.

| [root@node1 ~]# yum install pcs fence-agents-all -y  Dependencies Resolved  ==========================================================================  Package Arch Version Repository  Size  ==========================================================================  Installing:  fence-agents-all x86\_64 4.2.1-24.el7 base 23 k  pcs x86\_64 0.9.167-3.el7.centos.1 updates 4.1 M  Installing for dependencies:  OpenIPMI x86\_64 2.0.27-1.el7 base 243 k  OpenIPMI-libs x86\_64 2.0.27-1.el7 base 523 k  OpenIPMI-modalias x86\_64 2.0.27-1.el7 base 16 k  autogen-libopts x86\_64 5.18-5.el7 base 66 k  avahi-libs x86\_64 0.6.31-19.el7 base 61 k  bc x86\_64 1.06.95-13.el7 base 115 k  cifs-utils x86\_64 6.2-10.el7 base 85 k  clufter-bin x86\_64 0.77.1-1.el7 base 25 k  clufter-common noarch 0.77.1-1.el7 base 72 k  corosync x86\_64 2.4.3-6.el7\_7.1 updates 221 k  corosynclib x86\_64 2.4.3-6.el7\_7.1 updates 133 k  cups-libs x86\_64 1:1.6.3-40.el7 base 358 k  device-mapper-multipath x86\_64 0.4.9-127.el7 base 143 k  device-mapper-multipath-libs  x86\_64 0.4.9-127.el7 base 265 k  fence-agents-amt-ws x86\_64 4.2.1-24.el7 base 27 k  fence-agents-apc x86\_64 4.2.1-24.el7 base 27 k  fence-agents-apc-snmp x86\_64 4.2.1-24.el7 base 27 k  fence-agents-bladecenter x86\_64 4.2.1-24.el7 base 26 k  fence-agents-brocade x86\_64 4.2.1-24.el7 base 26 k  fence-agents-cisco-mds x86\_64 4.2.1-24.el7 base 26 k  fence-agents-cisco-ucs x86\_64 4.2.1-24.el7 base 26 k  fence-agents-common x86\_64 4.2.1-24.el7 base 76 k  fence-agents-compute x86\_64 4.2.1-24.el7 base 33 k  fence-agents-drac5 x86\_64 4.2.1-24.el7 base 26 k  fence-agents-eaton-snmp x86\_64 4.2.1-24.el7 base 27 k  fence-agents-emerson x86\_64 4.2.1-24.el7 base 25 k  fence-agents-eps x86\_64 4.2.1-24.el7 base 26 k  fence-agents-heuristics-ping  x86\_64 4.2.1-24.el7 base 26 k  fence-agents-hpblade x86\_64 4.2.1-24.el7 base 26 k  fence-agents-ibmblade x86\_64 4.2.1-24.el7 base 26 k  fence-agents-ifmib x86\_64 4.2.1-24.el7 base 26 k  fence-agents-ilo-moonshot x86\_64 4.2.1-24.el7 base 25 k  fence-agents-ilo-mp x86\_64 4.2.1-24.el7 base 25 k  fence-agents-ilo-ssh x86\_64 4.2.1-24.el7 base 30 k  fence-agents-ilo2 x86\_64 4.2.1-24.el7 base 28 k  fence-agents-intelmodular x86\_64 4.2.1-24.el7 base 26 k  fence-agents-ipdu x86\_64 4.2.1-24.el7 base 26 k  fence-agents-ipmilan x86\_64 4.2.1-24.el7 base 36 k  fence-agents-kdump x86\_64 4.2.1-24.el7 base 37 k  fence-agents-mpath x86\_64 4.2.1-24.el7 base 28 k  fence-agents-redfish x86\_64 4.2.1-24.el7 base 26 k  fence-agents-rhevm x86\_64 4.2.1-24.el7 base 27 k  fence-agents-rsa x86\_64 4.2.1-24.el7 base 25 k  fence-agents-rsb x86\_64 4.2.1-24.el7 base 25 k  fence-agents-sbd x86\_64 4.2.1-24.el7 base 27 k  fence-agents-scsi x86\_64 4.2.1-24.el7 base 30 k  fence-agents-vmware-rest x86\_64 4.2.1-24.el7 base 26 k  fence-agents-vmware-soap x86\_64 4.2.1-24.el7 base 28 k  fence-agents-wti x86\_64 4.2.1-24.el7 base 27 k  fence-virt x86\_64 0.3.2-14.el7 base 41 k  fontpackages-filesystem noarch 1.44-8.el7 base 9.9 k  gnutls x86\_64 3.3.29-9.el7\_6 base 680 k  gnutls-dane x86\_64 3.3.29-9.el7\_6 base 36 k  gnutls-utils x86\_64 3.3.29-9.el7\_6 base 238 k  gssproxy x86\_64 0.7.0-26.el7 base 110 k  ipmitool x86\_64 1.8.18-7.el7 base 441 k  keyutils x86\_64 1.5.8-3.el7 base 54 k  libbasicobjects x86\_64 0.1.1-32.el7 base 26 k  libcollection x86\_64 0.7.0-32.el7 base 42 k  liberation-fonts-common noarch 1:1.07.2-16.el7 base 27 k  liberation-sans-fonts noarch 1:1.07.2-16.el7 base 279 k  libevent x86\_64 2.0.21-4.el7 base 214 k  libini\_config x86\_64 1.3.1-32.el7 base 64 k  libldb x86\_64 1.4.2-1.el7 base 144 k  libnfsidmap x86\_64 0.25-19.el7 base 50 k  libpath\_utils x86\_64 0.2.1-32.el7 base 28 k  libqb x86\_64 1.0.1-7.el7 base 96 k  libref\_array x86\_64 0.1.5-32.el7 base 27 k  libtalloc x86\_64 2.1.14-1.el7 base 32 k  libtdb x86\_64 1.3.16-1.el7 base 48 k  libtevent x86\_64 0.9.37-1.el7 base 40 k  libtirpc x86\_64 0.2.4-0.16.el7 base 89 k  libverto-libevent x86\_64 0.2.5-4.el7 base 8.9 k  libwbclient x86\_64 4.9.1-6.el7 base 110 k  libwsman1 x86\_64 2.6.3-6.git4391e5c.el7\_6 base 139 k  libxslt x86\_64 1.1.28-5.el7 base 242 k  libyaml x86\_64 0.1.4-11.el7\_0 base 55 k  net-snmp-libs x86\_64 1:5.7.2-43.el7 base 750 k  net-snmp-utils x86\_64 1:5.7.2-43.el7 base 199 k  net-tools x86\_64 2.0-0.25.20131004git.el7 base 306 k  nettle x86\_64 2.7.1-8.el7 base 327 k  nfs-utils x86\_64 1:1.3.0-0.65.el7 base 412 k  openwsman-python x86\_64 2.6.3-6.git4391e5c.el7\_6 base 110 k  overpass-fonts noarch 2.1-1.el7 base 700 k  pacemaker x86\_64 1.1.20-5.el7\_7.1 updates 472 k  pacemaker-cli x86\_64 1.1.20-5.el7\_7.1 updates 360 k  pacemaker-cluster-libs x86\_64 1.1.20-5.el7\_7.1 updates 162 k  pacemaker-libs x86\_64 1.1.20-5.el7\_7.1 updates 629 k  perl-TimeDate noarch 1:2.30-2.el7 base 52 k  pexpect noarch 2.3-11.el7 base 142 k  psmisc x86\_64 22.20-16.el7 base 141 k  python-backports x86\_64 1.0-8.el7 base 5.8 k  python-backports-ssl\_match\_hostname  noarch 3.5.0.1-1.el7 base 13 k  python-chardet noarch 2.2.1-3.el7 base 227 k  python-clufter noarch 0.77.1-1.el7 base 323 k  python-ipaddress noarch 1.0.16-2.el7 base 34 k  python-lxml x86\_64 3.2.1-4.el7 base 758 k  python-requests noarch 2.6.0-7.el7\_7 updates 95 k  python-setuptools noarch 0.9.8-7.el7 base 397 k  python-six noarch 1.9.0-2.el7 base 29 k  python-suds noarch 0.4.1-5.el7 base 204 k  python-urllib3 noarch 1.10.2-7.el7 base 103 k  quota x86\_64 1:4.01-19.el7 base 179 k  quota-nls noarch 1:4.01-19.el7 base 90 k  resource-agents x86\_64 4.1.1-30.el7\_7.4 updates 453 k  rpcbind x86\_64 0.2.0-48.el7 base 60 k  ruby x86\_64 2.0.0.648-36.el7 base 73 k  ruby-irb noarch 2.0.0.648-36.el7 base 94 k  ruby-libs x86\_64 2.0.0.648-36.el7 base 2.8 M  rubygem-bigdecimal x86\_64 1.2.0-36.el7 base 85 k  rubygem-io-console x86\_64 0.4.2-36.el7 base 56 k  rubygem-json x86\_64 1.7.7-36.el7 base 81 k  rubygem-psych x86\_64 2.0.0-36.el7 base 84 k  rubygem-rdoc noarch 4.0.0-36.el7 base 324 k  rubygems noarch 2.0.14.1-36.el7 base 215 k  samba-client-libs x86\_64 4.9.1-6.el7 base 4.9 M  samba-common noarch 4.9.1-6.el7 base 209 k  samba-common-libs x86\_64 4.9.1-6.el7 base 170 k  sg3\_utils x86\_64 1.37-18.el7\_7.1 updates 646 k  sg3\_utils-libs x86\_64 1.37-18.el7\_7.1 updates 65 k  tcp\_wrappers x86\_64 7.6-77.el7 base 78 k  telnet x86\_64 1:0.17-64.el7 base 64 k  trousers x86\_64 0.3.14-2.el7 base 289 k  unbound-libs x86\_64 1.6.6-1.el7 base 405 k  Updating for dependencies:  kpartx x86\_64 0.4.9-127.el7 base 78 k  Transaction Summary  ========================================================================== |
| --- |

Теперь разрешаем взаимодействие между нодами в межсетевом экране.

| [root@node1 ~]# firewall-cmd --permanent --add-service=high-availability  success  [root@node1 ~]# firewall-cmd --reload  success |
| --- |

В результате установки у нас появился пользователь hacluster. Он отвечает за коммуникацию и конфигурацию кластера. Чтобы упростить последующее обслуживание кластера, имеет смысл сделать пароль для пользователя hacluster одинаковым на всех нодах. Зададим пароль centos:

| [root@node1 ~]# id hacluster  uid=189(hacluster) gid=189(haclient) groups=189(haclient)  [root@node1 ~]# echo centos | passwd --stdin hacluster  Changing password for user hacluster.  passwd: all authentication tokens updated successfully. |
| --- |

PCSD-сервис требует аутентификации каждой из нод при добавлении их в кластер. В этом примере используются IP-адреса, но в реальной жизни вам следует использовать доменные имена.

| [root@node1 ~]# systemctl enable pcsd  Created symlink from /etc/systemd/system/multi-user.target.wants/pcsd.service to /usr/lib/systemd/system/pcsd.service.  [root@node1 ~]# systemctl start pcsd  [root@node1 ~]# [root@node1 ~]# systemctl status pcsd  ● pcsd.service - PCS GUI and remote configuration interface  Loaded: loaded (/usr/lib/systemd/system/pcsd.service; enabled; vendor preset: disabled)  Active: active (running) since Sun 2019-11-24 11:14:34 MSK; 37s ago  Docs: man:pcsd(8)  man:pcs(8)  Main PID: 11283 (pcsd)  CGroup: /system.slice/pcsd.service  └─11283 /usr/bin/ruby /usr/lib/pcsd/pcsd  Nov 24 11:14:33 node1 systemd[1]: Starting PCS GUI and remote configur....  Nov 24 11:14:34 node1 systemd[1]: Started PCS GUI and remote configura....  Hint: Some lines were ellipsized, use -l to show in full.  [root@node1 ~]# pcs cluster auth 192.168.1.34 192.168.1.55 192.168.1.64  Username: hacluster  Password:  192.168.1.55: Authorized  192.168.1.34: Authorized  192.168.1.64: Authorized |
| --- |

После того, как ноды были авторизованы, мы можем создать и запустить кластер. Все настройки кластера с этого момента можно делать на одной ноде, они будут автоматически применяться на всех указанных членов кластера. В нашем случае это 3 ноды (192.168.1.34, 192.168.1.55, 192.168.1.64).

| [root@node1 ~]# pcs cluster setup --start --name mycluster 192.168.1.34 192.168.1.55 192.168.1.64  Destroying cluster on nodes: 192.168.1.34, 192.168.1.55, 192.168.1.64...  192.168.1.34: Stopping Cluster (pacemaker)...  192.168.1.64: Stopping Cluster (pacemaker)...  192.168.1.55: Stopping Cluster (pacemaker)...  192.168.1.34: Successfully destroyed cluster  192.168.1.64: Successfully destroyed cluster  192.168.1.55: Successfully destroyed cluster  Sending 'pacemaker\_remote authkey' to '192.168.1.34', '192.168.1.55', '192.168.1.64'  192.168.1.34: successful distribution of the file 'pacemaker\_remote authkey'  192.168.1.55: successful distribution of the file 'pacemaker\_remote authkey'  192.168.1.64: successful distribution of the file 'pacemaker\_remote authkey'  Sending cluster config files to the nodes...  192.168.1.34: Succeeded  192.168.1.55: Succeeded  192.168.1.64: Succeeded  Starting cluster on nodes: 192.168.1.34, 192.168.1.55, 192.168.1.64...  192.168.1.34: Starting Cluster (corosync)...  192.168.1.55: Starting Cluster (corosync)...  192.168.1.64: Starting Cluster (corosync)...  192.168.1.55: Starting Cluster (pacemaker)...  192.168.1.34: Starting Cluster (pacemaker)...  192.168.1.64: Starting Cluster (pacemaker)...  Synchronizing pcsd certificates on nodes 192.168.1.34, 192.168.1.55, 192.168.1.64...  192.168.1.55: Success  192.168.1.34: Success  192.168.1.64: Success  Restarting pcsd on the nodes in order to reload the certificates...  192.168.1.55: Success  192.168.1.34: Success  192.168.1.64: Success |
| --- |

Стандартно, если нода была перезагружена, она не будет добавлена обратно в кластер сама. Мы можем включить эту опцию, например, на всех сленов кластера.

| [root@node1 ~]# pcs cluster enable --all  192.168.1.34: Cluster Enabled  192.168.1.55: Cluster Enabled  192.168.1.64: Cluster Enabled |
| --- |

Проверить статус мы можем при помощи pcs cluster status.

| [root@node1 ~]# pcs cluster status  Cluster Status:  Stack: corosync  Current DC: node1 (version 1.1.20-5.el7\_7.1-3c4c782f70) - partition with quorum  Last updated: Sun Nov 24 11:22:06 2019  Last change: Sun Nov 24 11:19:06 2019 by hacluster via crmd on node1  3 nodes configured  0 resources configured  PCSD Status:  node3 (192.168.1.64): Online  node1 (192.168.1.34): Online  node2 (192.168.1.55): Online |
| --- |

Если бы мы говорили про настоящий кластер, сейчас бы настал момент настроить fencing-агент, чтобы предотвратить потерю данных в случае, если одна из нод кластера функционирует неправильно. Fencing-агент поддерживает интеграцию с различными моделями серверов, устройств бесперебойного питания, гипервизоров для виртуальных машин и так далее. Настраивается это при помощи команд pcs stonith. Посмотреть весь список и возможные опции вы можете так:

| [root@node1 ~]# pcs stonith --help |
| --- |

Мы сейчас это настроить не можем, так как интеграция с вашей системой виртуализации может отсутствовать.

PCS позволяет управлять как самими сервисами кластера, так и нодами в кластере. То есть можно легко вывести ноду для обслуживания из кластера, а потом добавить её обратно.

| [root@node1 ~]# pcs cluster stop 192.168.1.64  192.168.1.64: Stopping Cluster (pacemaker)...  192.168.1.64: Stopping Cluster (corosync)... |
| --- |

# Кворум

Чтобы обслуживать пользователей, ноды кластера должны договориться о некоторых параметрах: кто именно обслуживает клиентов, работают ли сервисы как положено, какие ноды контролируют ресурсы и так далее. Чтобы сделать это гарантированно надёжно, большинство нод в кластере должно иметь одинаковую точку зрения. Такое большинство называется кворумом и каждая нода в кластере имеет 1 голос. Кластер считается работоспособным, если кворум достигнут. Если кластер теряет кворум, он считается неработоспособным.

При старте кластера ноды начинают взаимодействовать друг с другом для достижения кворума. Как только большинство собралось и кворум достигнут, ноды, которые не согласны или не смогли договориться с большинством, отключаются при помощи fencing.

Кворум важен в ситуациях, когда некоторые ноды из-за проблем, например, с сетью, не в состоянии коммуницировать с оставшимися нодами кластера. В такой ситуации, кластер разделён на несколько частей, но кластерные ресурсы должны контролироваться кем-то одним — большинством.

Проверить кворум кластера можно при помощи утилиты corosync-quorumtool.

| [root@node1 ~]# corosync-quorumtool  Quorum information  ------------------  Date: Mon Nov 24 23:42:25 2019  Quorum provider: corosync\_votequorum  Nodes: 3  Node ID: 1  Ring ID: 1/20  Quorate: Yes  Votequorum information  ----------------------  Expected votes: 3  Highest expected: 3  Total votes: 3  Quorum: 2  Flags: Quorate  Membership information  ----------------------  Nodeid Votes Name  1 1 192.168.1.34 (local)  2 1 192.168.1.55  3 1 192.168.1.64 |
| --- |

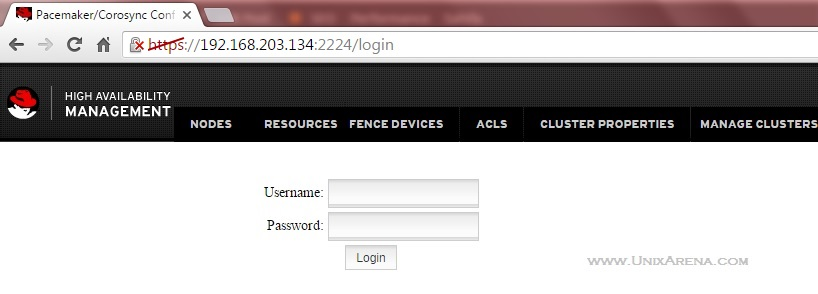
1. Nodes — общее количество нод в кластере.
2. Node ID — ID ноды, на которой была выполнена утилита corosync-quorumtool.
3. Quorate показывает, если кворум присутствует.
4. Expected votes показывает количество голосов, если все ноды в кластере функционируют.

Параметры кворума можно задать при конфигурировании кластера, либо модифицировав файл конфигурации /etc/corosync/corosync.conf.

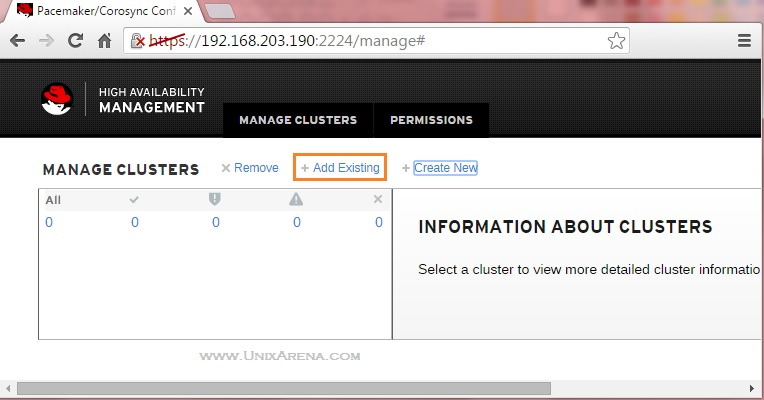
* --auto\_tie\_breaker При включении кластер может пережить потерю до 50% узлов, вышедших из строя одновременно. Та часть кластера, которая содержит ноду с NODE ID, настроенным в auto\_tie\_breaker\_node (или самым низким NODE ID, если этот параметр не задан), будет обладать кворумом. Вторая же половина будет отключена.
* Параметр auto\_tie\_breaker в основном используется для кластеров с чётным числом узлов, поскольку он позволяет кластеру продолжать работу с отключением ровно половины нод.
* --wait\_for\_all. Кворум будет собран только в случае, если все ноды доступны одновременно.
* Опция wait\_for\_all в основном используется для кластеров с двумя нодами или для кластеров с чётным узлом и использованием алгоритма кворума lms (last man standing).
* --last\_man\_standing. Когда этот параметр включен, кластер может динамически пересчитывать ожидаемые голоса и кворум при определённых обстоятельствах. Вы должны включить wait\_for\_all при включении этой опции.

| [root@node1 ~]# cat /etc/corosync/corosync.conf  totem {  version: 2  cluster\_name: mycluster  secauth: off  transport: udpu  }  nodelist {  node {  ring0\_addr: 192.168.1.34  nodeid: 1  }  node {  ring0\_addr: 192.168.1.55  nodeid: 2  }  node {  ring0\_addr: 192.168.1.64  nodeid: 3  }  }  quorum {  provider: corosync\_votequorum  }  logging {  to\_logfile: yes  logfile: /var/log/cluster/corosync.log  to\_syslog: yes  } |
| --- |

Запустив кластер, дальнейшее управление им можно осуществлять через GUI. Стандартный для этого порт — tcp 2224. Пользователь — hacluster с паролем, который мы задали ранее.



Введя логин и пароль, мы попадаем в графический интерфейс управления кластером, в который надо добавить уже существующий кластер.



После чего можно запускать нужные нам кластерные сервисы.

# Практическое задание

1. Развернуть 3 виртуальные машины и настроить HA-кластер на них.
2. Развернуть на этом кластере высокодоступный веб-сервер Apache.

# Дополнительные материалы

1. [Red Hat Enterprise Linux Cluster, High Availability, and GFS Deployment Recommended Practices](https://access.redhat.com/articles/40051).
2. [rhel High Availability and Pacemaker 101!](http://www.juliosblog.com/pacemaker-101-2/)

# Используемая литература

1. [Product Documentation for Red Hat Enterprise Linux 7](https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/).