Отказоустойчивый кластер Платформы НЕЙРОСС

Общие сведения

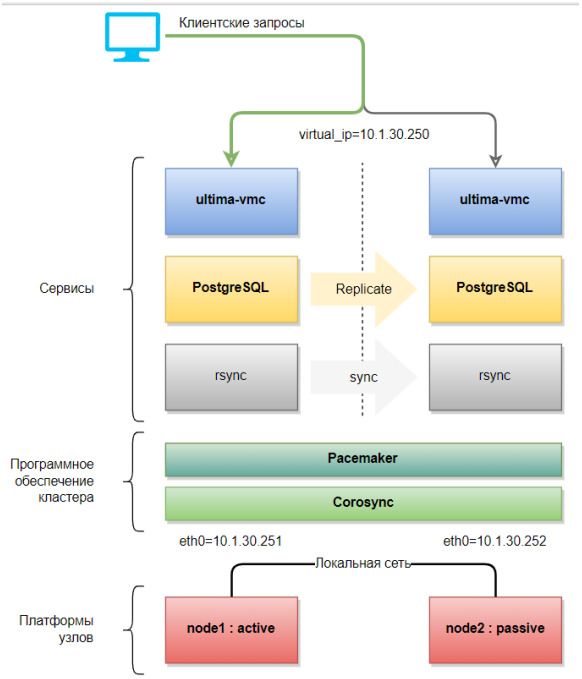
Содержание:

Отказоустойчивый кластер обеспечивает высокий уровень готовности резервного сервера для максимальной доступности сервисов системы.

Самый простой кластер состоит из 2-х узлов (компьютеров, серверов), образующих его, настроенных на мониторинг друг друга и управление соответствующими ресурсами кластера.

В данном разделе будет рассмотрена пошаговая инструкция «с нуля» по развёртыванию кластера, обеспечивающего базовую функциональность горячего резервирования с учётом прикладного применения с Платформой НЕЙРОСС.

Кластер разворачивается на ОС Ubuntu Server 20.04.3 LTS.



Для настройки кластера и его ресурсов можно применять любую из двух утилит *pcs* или *crm*. При этом настройку можно осуществлять и той и другой в любом удобном порядке, поэтому некоторые действия описаны с использование одной утилиты, а некоторые – другой.

В данном разделе не рассматривается развёртывание собственно ОС и программных средств Платформы НЕЙРОСС.

Развёртывание и настройка кластера

Действия на каждом сервере

Актуализируем и обновим версии пакетов:

```
sudo apt update
sudo apt upgrade
```

Проинсталлируем программные средства, необходимые для работы платформы НЕЙРОСС (на момент написания страницы из репозитория инсталлировался PostgreSQL версии 12):

```
sudo apt install postgresql
sudo apt install -y openjdk-8-jdk traceroute
```

Проинсталлируем пакеты *pacemaker* и *corosync*, а также соответствующие необходимые утилиты. Рекомендуется выполнять установку из-под root для правильного создания пользователя *hacluster* и настройки пользовательского доступа.

```
sudo su
apt install pacemaker pcs resource-agents fence-agents corosync ntp rsync
exit
```

Указываем в файле *hosts* IP-адреса всех узлов кластера в явном виде:

```
sudo nano /etc/hosts
  10.1.30.251 node1
  10.1.30.252 node2
```

Настраиваем службу синхронизации времени на сервер точного времени, перезапускаем службу, проверяем работу:

```
sudo nano /etc/ntp.conf
sudo systemctl restart ntp
sudo ntpq -p
```

Проверяем пользователя *hacluster* (его создает *pacemaker* в процессе инсталляции):

```
sudo cat /etc/passwd | grep hacluster
```

вывод:

```
hacluster:x:113:117::/var/lib/pacemaker:/usr/sbin/nologin
```

и меняем ему пароль (например, 123456)

```
sudo passwd hacluster
  123456
  123456
```

⚠ Пакет *resource-agents,* инсталлируемый из стандартного репозитория, имеет версию ниже 4.8.0, и значит в нём не исправлена несовместимость с PostgreSQL 12 в части мониторинга 'WAL receiver process'.
Решением является установка пакета версии 4.8.0 и выше или исправление соответствующего файла руками (на обоих серверах). На момент написания этой страницы пакет найти для Ubuntu мне не удалось, поэтому исправляем файл:

```
sudo nano /usr/lib/ocf/resource.d/heartbeat/pgsql
```

ищем следующую строку:

```
receiver_parent_pids=`ps -ef | tr -s " " | grep "[w]al receiver process" | cut -d " " -f 3`
```

меняем её на:

```
receiver_parent_pids=`ps -ef | tr -s " " | grep "[w]al\s*receiver" | cut -d " " -f 3`
```

выполняем сохранение (^O) и выход (^X).

Теперь добавляем в автозагрузку и запускаем службу конфигурации *pacemaker:*

```
sudo systemctl enable pcsd.service
sudo systemctl start pcsd.service
```

Действия на первом узле (node1) будущего кластера

Определяем авторизацию на узлах (имена узлов *node1* и *node2*) под пользователем *hacluster*:

```
sudo pcs cluster auth node1 addr=10.1.30.251 node2 addr=10.1.30.252 -u hacluster -p 123456
```

Если команда не выполнилась (это может зависеть от версии *pacemaker*), то выполним:

```
sudo pcs host auth node1 addr=10.1.30.251 node2 addr=10.1.30.252 -u hacluster -p 123456
```

Создаем кластер с именем *HACLUSTER* из двух узлов:

```
sudo pcs cluster setup HACLUSTER node1 addr=10.1.30.251 node2 addr=10.1.30.252
```

Если возникают ошибки с текстом "...*the host seems to be in a cluster already*...", то необходимо выполнить:

```
sudo pcs cluster setup --force HACLUSTER node1 addr=10.1.30.251 node2 addr=10.1.30.252
```

При необходимости проверить конфигурацию (на всех серверах должен быть файл с одинаковым содержимым) выполняем:

```
cat /etc/corosync/corosync.conf
        totem {
            version: 2
            cluster_name: HACLUSTER
            transport: knet
            crypto_cipher: aes256
            crypto_hash: sha256
        }

        nodelist {
            node {
            ring0_addr: 10.1.30.251
            name: node1
            nodeid: 1
            }

            node {
            ring0_addr: 10.1.30.252
            name: node2
            nodeid: 2
            }
        }

        quorum {
            provider: corosync_votequorum
            two_node: 1
        }

        logging {
            to_logfile: yes
            logfile: /var/log/corosync/corosync.log
            to_syslog: yes
            timestamp: on
        }
```

Включаем и запускаем все кластеры на всех узлах:

```
sudo pcs cluster enable --all
sudo pcs cluster start --all
```

При использовании двух узлов включаем *stonith*. Он нужен для «добивания» серверов, которые не смогли полностью завершить рабочие процессы, игнорируем кворум:

```
sudo pcs property set stonith-enabled=true
sudo pcs property set no-quorum-policy=ignore
```

Без сконфигурированного *stonith* кластер не начнёт управлять ресурсами. Поэтому, в этом месте для простоты старта работы кластера сначала можно выключить *stonith:*

```
sudo pcs property set stonith-enabled=false
```

Потом, когда будет сконфигурирован *stonith*, включить его обратно (описано ниже) для обеспечения фенсинга.

Запрашиваем статус на обоих узлах:

```
sudo pcs status
```

Видим (в случае *stonith-enabled=true*):

```
Cluster name: HACLUSTER

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync
  * Current DC: node1 (version 2.0.3-4b1f869f0f) - partition with quorum
  * Last updated: Wed Oct 20 14:06:00 2021
  * Last change:  Wed Oct 20 14:05:17 2021 by root via cibadmin on node1
  * 2 nodes configured
  * 0 resource instances configured

Node List:
  * Online: [ node1 node2 ]

Full List of Resources:
  * No resources

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

Для мониторинга (отслеживания) состояния кластера в реальном времени можно использовать команду на любом из узлов:

```
sudo crm_mon -Afr
```

Добавляем виртуальный сетевой адрес как ресурс (помним про тайм-ауты) кластера с именем *virtual_ip*, который и будет основным адресом платформы НЕЙРОСС:

```
sudo pcs resource create virtual_ip ocf:heartbeat:IPaddr2 ip=10.1.30.250
cidr_netmask=13 meta migration-threshold="0" \
op monitor timeout="60s" interval="10s" on-fail="restart" \
op stop timeout="60s" interval="0s" on-fail="ignore" \
op start timeout="60s" interval="0s" on-fail="stop"
```

Отключаем запуск *postgresql.service* при загрузке системы. Включать и отключать сервис при необходимости теперь будет *pacemaker*.

```
sudo systemctl disable postgresql.service
```

На узле (в нашем случае это node1), который первоначально будет являться Мастером, инициализируйте новую базу данных:

```
sudo -u postgres /usr/lib/postgresql/12/bin/initdb -D /var/lib/postgresql/12/main
```

Если база уже запущена, то останавливаем процесс *postgresql*, очищаем директорию */var/lib/postgresql/12/main*

```
sudo systemctl stop postgresql.service
sudo su - postgres
rm -rf /var/lib/postgresql/12/main/*
```

и выполняем команду инициализации новой базы ещё раз.

Запускаем базу:

```
sudo -u postgres /usr/lib/postgresql/12/bin/pg_ctl -D /var/lib/postgresql/12/main start
```

Создаём пользователя для репликации базы:

```
sudo -u postgres /usr/lib/postgresql/12/bin/createuser --replication -P repl
```

Устанавливаем пароль, например, *12345.*

Изменяем файл */var/lib/postgresql/12/main/pg_hba.conf:*

```
sudo -u postgres nano /var/lib/postgresql/12/main/pg_hba.conf
```

Добавляем в него необходимые разрешения следующим образом:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
host    replication     all             10.0.0.0/13             trust
host    all             all             10.0.0.0/13             trust
```

Изменяем файл */var/lib/postgresql/12/main/postgresql.conf:*

```
sudo -u postgres nano /var/lib/postgresql/12/main/postgresql.conf
```

Добавляем (или раскомментируем) в него следующие строки:

```
listen_addresses = '*'
wal_level = replica
logging_collector = on
hot_standby = on
wal_keep_segments = 10
```

Перезапускаем СУБД PostgreSQL:

```
sudo -u postgres /usr/lib/postgresql/12/bin/pg_ctl -D /var/lib/postgresql/12/main stop
sudo -u postgres /usr/lib/postgresql/12/bin/pg_ctl -D /var/lib/postgresql/12/main start
```

Тут инсталлируем Платформу НЕЙРОСС на этом узле и проходим "Первый запуск" по адресу *10.1.30.250*, во избежание в момент перезагрузки узла *node1* перехвата общего адреса узлом *node2* перед выполнением сохранения параметров и перезапуска узла *node1* останавливаем узел *node2* и запускаем только после начала включения узла *node1*.

На другом узле (*node2*) кластера выполняем следующие действия

Отключаем запуск *postgresql.service* при загрузке системы. Включать и отключать сервис при необходимости теперь будет *pacemaker*.

```
sudo systemctl disable postgresql.service
```

Останавливаем процесс *postgresql*, если он ещё запущен.
Очищаем директорию */var/lib/postgresql/12/main/:*

```
sudo systemctl stop postgresql.service
sudo su - postgres
rm -rf /var/lib/postgresql/12/main/*
```

Скопируем базу данных с Мастера (*node1*) при помощи команды:

```
pg_basebackup -U postgres -D /var/lib/postgresql/12/main -h 10.1.30.251 -X stream -P
exit
```

В результате в директории */var/lib/postgresql/12/main* узла *node2* появится содержимое директории */var/lib/postgresql/12/main* узла *node1.*

Тут инсталлируем Платформу НЕЙРОСС на этом узле и не проходим "Первый запуск".
Отключаем запуск *ultima-vmc.service* при загрузке системы на этом узле кластера. Включать и отключать сервис при необходимости теперь будет *pacemaker*.

```
sudo systemctl stop ultima-vmc.service
sudo systemctl disable ultima-vmc.service
```

Продолжаем на узле (node1) и выполняем действия

Создаем ресурс с именем *HA-pgsql* типа *pgsql* для управления конфигурацией PostgreSQL:

```
sudo pcs resource create HA-pgsql pgsql pgctl="/usr/lib/postgresql/12/bin/pg_ctl" \
psql="/usr/lib/postgresql/12/bin/psql" \
pgdata="/var/lib/postgresql/12/main" rep_mode="sync" \
node_list="node1 node2" master_ip="10.1.30.250" \
restart_on_promote="false" check_wal_receiver="false" pgport="5432" \
primary_conninfo_opt="password=12345" repuser="repl" check_wal_receiver="true"
```

В некоторых описаниях настройки упоминается ещё параметр config="/etc /postgresql/12/main/postgresql.conf".

Если не выполнено исправление *resource-agents* (описано выше), то настройка параметра *check_wal_receiver="true"*, приведёт на узле *node2* (Slave) к появлению ошибки в *HA-pgsql-receiver-status* (вывод команды *sudo crm_mon -Afr*) и ежеминутных предупреждений в логах (*journalctl -f*) "WARNING: wal receiver process is not running"

Для созданного выше ресурса *HA-pgsql* укажем, что он может иметь одно из нескольких состояний и менять их в зависимости от типа узла (master и slave):

```
sudo pcs resource promotable HA-pgsql promoted-max=1 promoted-node-max=1
clone-max=2 clone-node-max=1 notify=true
```

Свяжем два созданных выше ресурса (*HA-pgsql* и *virtual_ip*), чтобы они запускались вместе на одном узле, и установим очерёдность запуска таким образом, чтобы ресурс *virtual_ip* запускался только после успешного запуска ресурса *HA-pgsql*. Для этого создаем группу ресурсов *master-group* и добавляем в неё ресурсы:

```
sudo pcs resource group add master-group virtual_ip
sudo pcs constraint colocation add master-group with Master HA-pgsql-clone
sudo pcs constraint order promote HA-pgsql-clone then start master-group
symmetrical=false kind=Mandatory
sudo pcs constraint order demote HA-pgsql-clone then stop master-group
symmetrical=false kind=Optional
```

Отключаем запуск *ultima-vmc.service* при загрузке системы на этом узле кластера. Включать и отключать сервис при необходимости теперь будет *pacemaker*.

```
sudo systemctl stop ultima-vmc.service
sudo systemctl disable ultima-vmc.service
```

Копируем файл *application.conf* с узла *node1* на узел *node2*, для этого на *node1* выполним:

```
scp /usr/share/ultima-vmc/conf/application.conf user@10.1.30.252:/usr/share/ultima-vmc/conf/application.conf
```

Копируем содержимое каталога */home/ultima-vmc/* с узла *node1* на узел *node2*, для этого на *node1* выполним:

```
sudo scp -r /home/ultima-vmc/* user@10.1.30.252:/home/ultima-vmc/
```

На узле node2

Устанавливаем в явном виде владельца для скопированных файлов, для этого на *node2*:

```
sudo chown -R ultima-vmc:ultima-vmc /home/ultima-vmc/
sudo chown ultima-vmc:ultima-vmc /usr/share/ultima-vmc/conf/application.conf
```

На узле node1

Создаем ресурс с именем *HA-neyross* типа *systemd* для управления конфигурацией исполняемого сервиса и сразу добавим созданный ресурс в группу *master-group*, чтобы он запускался вместе на одном узле с другими ресурсами:

```
sudo pcs resource create HA-neyross systemd:ultima-vmc.service \
op monitor OCF_CHECK_LEVEL="0" timeout="20s" interval="10s" \
--group master-group
```

или, для нашего случая менее подходящее решение, создаем ресурс с именем HA-neyross типа anything для управления конфигурацией исполняемой программы:

```
sudo pcs resource create HA-neyross ocf:heartbeat:anything \
binfile=... cmdline_options=... user=... \
op monitor OCF_CHECK_LEVEL="0" timeout="20s" interval="10s" \
--group master-group
```

Заключение

Финально вывод команды мониторинга

```
sudo crm_mon -Afr
```

выглядит следующим образом:

```
Cluster Summary:
  * Stack: corosync
  * Current DC: node1 (version 2.0.3-4b1f869f0f) - partition with quorum
  * Last updated: Mon Nov  1 17:02:05 2021
  * Last change:  Mon Nov  1 16:54:44 2021 by root via crm_attribute on node1
  * 2 nodes configured
  * 6 resource instances configured

Node List:
  * Online: [ node1 node2 ]

Full List of Resources:
  * Clone Set: HA-pgsql-clone [HA-pgsql] (promotable):
    * Masters: [ node1 ]
    * Slaves: [ node2 ]
  * Resource Group: master-group:
    * virtual_ip        (ocf::heartbeat:IPaddr2):        Started node1
    * HA-neyross        (systemd:ultima-vmc.service):    Started node1
  * Clone Set: fencing [st-ssh]:
    * Started: [ node1 node2 ]

Node Attributes:
  * Node: node1:
    * HA-pgsql-data-status              : LATEST
    * HA-pgsql-master-baseline          : 00000000077D6EF8
    * HA-pgsql-receiver-status          : normal (master)
    * HA-pgsql-status                   : PRI
    * master-HA-pgsql                   : 1000
  * Node: node2:
    * HA-pgsql-data-status              : STREAMING|SYNC
    * HA-pgsql-receiver-status          : normal
    * HA-pgsql-status                   : HS:sync
    * master-HA-pgsql                   : 100

Migration Summary:
```

HA-pgsql-status
PRI – состояние мастера
HS:sync – синхронная реплика
HS:async – асинхронная реплика
HS:alone – реплика не может подключится к мастеру
STOP – PostgreSQL остановлен
HA-pgsql-data-status
LATEST – состояние, присущее мастеру. Данный узел является мастером.
STREAMING:SYNC/ASYNC – показывает состояние репликации и тип репликации (SYNC/ASYNC)
DISCONNECT – реплика не может подключиться к мастеру. Обычно такое бывает, когда нет соединения от реплики к мастеру.
HA-pgsql-master-baseline
Показывает линию времени. Линия времени меняется каждый раз после выполнения команды promote на узле-реплике. После этого СУБД начинает новый отсчет времени.
HA-pgsql-receiver-status
normal (master) – состояние, присущее мастеру. Данный узел является мастером.
normal – нормальное состояние, присущее узлу-реплике. На ведомом устройстве запущен и работает процесс приёмника WAL.
Error – на ведомом устройстве не работает процесс приёмника WAL или отсутствует коммуникация отправителя и приёмника WAL.

На этом базовая настройка отказоустойчивого кластера PostgreSQL и Платформы НЕЙРОСС окончена.

Полезные команды

Для мониторинга (отслеживания) состояния кластера в реальном времени можно использовать команду:

```
sudo crm_mon -Afr
```

Перевод узла в standby

```
sudo pcs node standby node2
```

Возврат узла из standby

```
sudo pcs node unstandby node2
```

Если PostgreSQL (ресурс HA-pgsql) остаётся в состоянии "Stopped" на узле node2 и "Failed Resource Actions" листинг выводит "error", выполните для диагностики:

```
sudo pcs resource debug-start postgresql
```

Если узел сообщит:
'My data may be inconsistent. You have to remove /var/lib/pgsql/tmp/PGSQL.lock file to force start.'
Необходимо удалить файл */var/lib/pgsql/tmp/PGSQL.lock* для возможности старта
Для удаления и очистки счётчика сбоев выполните:

```
sudo rm /var/lib/pgsql/tmp/PGSQL.lock
sudo pcs resource cleanup HA-pgsql
```

Проверка конфигурации:

```
sudo crm_verify -L -VVV
```

При отсутствии ошибок выводит:

```
(unpack_config)         notice: On loss of quorum: Ignore
```

Проверка сбоев у конкретного ресурса (например, *HA-pgsql*), которые препятствуют его старту:

```
sudo pcs resource failcount show HA-pgsql
```

Проверка сбоев у всех ресурсов, которые препятствуют их старту:

```
sudo pcs resource failcount show
```

Очистка счётчика всех сбоев (применяется после устранения причин сбоя):

```
sudo pcs resource cleanup
```

Очистка счётчика сбоев фенсинга узла *node2* (применяется после устранения причин сбоя):

```
sudo stonith_admin --cleanup --history=node2
```

crm ra classes
crm ra list systemd или sudo crm_resource --list-agents lsb или sudo crm_resource --list-agents ocf:heartbeat

---

Создание ресурсов фенсинга

Для защиты разделяемых ресурсов и изоляции узла кластера при его неисправности существует механизм фенсинга (изоляции).

Во избежание ситуации появления двух Мастеров (например, в следствии потери сетевой связанности между узлами) необходимо наличие устройств «фенсинга» на узлах с СУБД и сервисами. При возникновении сбоя такие устройства «фенсинга» изолируют «сбойнувший» узел – посылают команду на выключение питания или перезагрузку (*poweroff* или *hard-reset*).

Чтобы вывести список доступных агентов *fence-agents* используйте команду:

```
sudo pcs stonith list
```

Для тестовых целей можно использовать следующую конфигурацию с агентом *external/ssh*:

```
sudo crm
 configure
  primitive st-ssh stonith:external/ssh params hostlist="node1 node2"
  clone fencing st-ssh
  property stonith-enabled=true
  commit
 exit
```

Пример 1. Использование двух механизмов фенсинга узлов на примере фенсинга виртуальных машинах

При использовании виртуальных машин в качестве узлов кластера можно использовать агент *external/libvirt*. Далее рассмотрим настройку конфигурации, в которой сервер с гипервизором имеет адрес 10.1.30.249, а узлы, как описано выше. В примере будут использоваться два фенсинг механизма – *ssh* и *libvirt*. Чтобы вывести необходимые настройки для выбранного агента выполните команду:

```
sudo pcs stonith describe external/libvirt
```

Основным механизмом фенсинга виртуальных машин является агент *libvirt* (или vcenter, xen и т.д.), но в случае, если хост виртуальной машины (гипервизор) не работает, фенсинг через *libvirt* никогда не будет успешным.

Идея состоит в том, чтобы реализовать второй механизм фенсинга, например, IPMI, который сработает при выходе из строя первого механизма.

Для демонстрации идеи, в этом примере наоборот первым механизмом фенсинга будет агент *ssh* (фенсинг узла на виртуальной машине), а вторым механизмом будет агент *libvirt* (фенсинг виртуальной машины на хосте гипервизора). Таким образом, что если виртуальная машина (node1 или node2) зависла и не может управляться агентом *ssh*, то фенсинг будет осуществлён через *libvirt* и, соответственно, сервер с гипервизором, на котором эта машина работает.

Обменяемся ssh ключами между узлами кластера (виртуальными машинами). Для этого на узле *node1* сгенерируем ключ, передадим на узел *node2* и проверим сессию

```
ssh-keygen
ssh-copy-id user@node2
ssh user@node2
exit
```

аналогично на узле *node2*

```
ssh-keygen
ssh-copy-id user@node1
ssh user@node1
exit
```

Сконфигурируем ресурсы фенсинга

```
sudo crm
 confugure
  primitive fence-node1-libvirt stonith:external/libvirt \
        params hostlist=node1 hypervisor_uri="qemu+ssh://10.1.30.249/system"
reset_method=power_cycle \
        op monitor interval=180 timeout=30 \
        meta target-role=Started
  primitive fence-node1-ssh stonith:ssh \
        params hostlist=node1 stonith-timeout=30 \
        meta target-role=Started
  primitive fence-node2-libvirt stonith:external/libvirt \
        params hostlist=node2 hypervisor_uri="qemu+ssh://10.1.30.249/system"
reset_method=power_cycle \
        op monitor interval=180 timeout=30 \
        meta target-role=Started
  primitive fence-node2-ssh stonith:ssh \
        params hostlist=node2 stonith-timeout=30 \
        meta target-role=Started
  location l-fence-node1-libvirt fence-node1-libvirt -inf: node1
  location l-fence-node1-ssh fence-node1-ssh -inf: node1
  location l-fence-node2-libvirt fence-node2-libvirt -inf: node2
  location l-fence-node2-ssh fence-node2-ssh -inf: node2
  fencing_topology \
        node2: fence-node2-ssh fence-node2-libvirt \
        node1: fence-node1-ssh fence-node1-libvirt
  property cib-bootstrap-options: \
        stonith-enabled=yes \
        no-quorum-policy=ignore \
        placement-strategy=balanced \
        dc-version=1.1.12-ad083a8 \
        cluster-infrastructure=corosync \
        cluster-name=hacluster \
        stonith-timeout=90 \
        last-lrm-refresh=1420721144
  rsc_defaults rsc-options: \
        resource-stickiness=1 \
        migration-threshold=3
  op_defaults op-options: \
        timeout=600 \
        record-pending=true
  commit
 exit
```

Пояснение определения

```
fencing_topology \
  node2: fence-node2-ssh fence-node2-libvirt \
  node1: fence-node1-ssh fence-node1-libvirt
```

означает: для фенсинга узла *node2* сначала использовать ресурс *fence-node2-ssh*, если это не удается, то использовать ресурс *fence-node2-libvirt*.

Перезагружаем все виртуальные машины в кластере.

Пример 2. Фенсинг узлов адаптером удалённого супервизора (RSA)

Реальная конфигурация не сильно отличается от тестовой, хотя для некоторых фенсинг устройств может потребоваться больше атрибутов. Например, устройство отключения IBM RSA (например, с адресами 10.1.31.101 и 10.1.31.102) может быть настроено следующим образом:

```
sudo crm
configure
  primitive st-ibmrsa-1 stonith:external/ibmrsa-telnet \
        params nodename=node1 ipaddr=10.1.31.101 \
        userid=USERID passwd=PASSW0RD
  primitive st-ibmrsa-2 stonith:external/ibmrsa-telnet \
        params nodename=node2 ipaddr=10.1.31.102 \
        userid=USERID passwd=PASSW0RD
# st-ibmrsa-1 может работать где угодно, но не на узле node1
  location l-st-node1 st-ibmrsa-1 -inf: node1
# st-ibmrsa-2 может работать где угодно, но не на узле node2
  location l-st-node2 st-ibmrsa-2 -inf: node2
commit
```

Пример 3. Фенсинг узлов агентом источников бесперебойного питания APC PDU

Ниже приведен полный пример двухузлового кластера, в котором каждый сервер имеет один источник питания, подключенный к общему APC PDU на разные розетки:

```
sudo pcs stonith create node1-node2-power-apc stonith:apcmaster \
  ipaddr="10.1.31.11" \
  login="apc" \
  password="apc" \
  pcmk_host_list="node1,node2" \
  pcmk_host_check="static-list" \
  pcmk_host_map="node1:7;node2:8"
```

ipaddr – это IP-адрес контроллера APC PDU. Внимание, это не IP-адрес узла, который будет изолирован.
login и *password* используются для предоставления учетных данных для входа в контроллер APC PDU.
pcmk_host_map – сопоставляет имя узла в *pacemaker* с номером порта на PDU, представляющем физическую розетку ИБП APC. Каждая запись в списке имеет формат <имя узла>:<номер порта PDU> (двоеточие отделяет узел от порта), а записи между собой разделяются точкой с запятой.

Пример 4. Фенсинг узлов с резервными источниками питания и несколькими источниками бесперебойного питания APC PDU

Когда серверы имеют резервные источники питания с несколькими подключениями к источникам бесперебойного питания, важно, чтобы кластер *pacemaker* мог отключать питание всех блоков питания в сервере при попытке изолировать узел.
Для этого должно быть определение фенсинг агента для каждого PDU, который подаёт питание на серверы узлов в кластере.
В следующем примере определены два фенсинг агента APC:

```
sudo pcs stonith create node1-node2-power-apc1 stonith:apcmaster \
  ipaddr="10.1.30.11" \
  login="apc" \
  password="apc" \
  pcmk_host_list="node1,node2" \
  pcmk_host_check="static-list" \
  pcmk_host_map="node1:7;node2:8"

sudo pcs stonith create node1-node2-power-apc2 stonith:apcmaster \
  ipaddr="10.1.30.12" \
  login="apc" \
  password="apc" \
  pcmk_host_list="node1,node2" \
  pcmk_host_check="static-list" \
  pcmk_host_map="node1:7;node2:8"
```

В этом примере каждый сервер подключен к одному и тому же порту питания (физической розетке) на каждом из двух PDU. Это может быть не всегда, поэтому убедитесь, что *pcmk_host_map* отражает физическую конфигурацию каждого PDU.

Чтобы гарантировать, что все определенные порты питания (розетки) каждого PDU отключены одновременно, фенсинг агенты должны быть сгруппированы в уровень фенсинга. Уровень – это разделённый запятыми список фенсинг ресурсов, которые необходимо выполнить, чтобы изолировать (выключить сервер) узел кластера. Уровней может быть несколько, в зависимости от сложности кластера и количества доступных вариантов фенсинга. Каждый уровень является автономным, и выполнение фенсинга прекращается, когда все фенсинг агенты на данном уровне завершаются с успешным кодом выхода (завершения).

Если на уровне STONITH определено несколько агентов, все агенты должны успешно завершиться, хотя они не обязательно должны работать одновременно.

В продолжение этого примера, уровни STONITH можно определить следующим образом:

```
sudo pcs stonith level add 1 node1 \
node1-node2-power-apc1,node1-node2-power-apc2
sudo pcs stonith level add 1 node2 \
node1-node2-power-apc1,node1-node2-power-apc2
```

Пример 5. Фенсинг узлов на виртуальных машинах

При использовании виртуальных машин в качестве узлов кластера можно использовать агент *fence_virsh*. Далее рассмотрим настройку конфигурации, в которой сервер с гипервизором имеет адрес 10.1.30.249, а узлы, как описано выше.
Чтобы вывести необходимые настройки для выбранного агента используйте команду:

```
sudo pcs stonith describe fence_virsh
```

Настройка доступа по ssh

Чтобы настроить доступ по ssh к серверу с гипервизором под пользователем root по ключу выполните следующие действия.
На сервере в файле */etc/ssh/sshd_config* установите значение параметра *PermitRootLogin* равное *yes*.
Перезагрузите службу на сервере *sshd*:

```
sudo systemctl restart sshd.service
```

На каждом узле сгенерируйте ключи при помощи команды:

```
sudo ssh-keygen
```

На каждом узле отправьте публичный ключ на сервер с гипервизором (например, адрес сервера гипервизора *10.1.30.249*):

```
sudo ssh-copy-id root@10.1.30.249
```

На сервере в файле */etc/ssh/sshd_config* закомментируйте параметр *PermitRootLogin* (чтобы он не применялся в конфигурации).
Перезагрузите на сервере службу *sshd* для применения настроек:

```
sudo systemctl restart sshd.service
```

Для проверки работы *fence_virsh* перед настройкой можно использовать команду:

```
sudo fence_virsh -a 10.1.30.249 -l root -n node1 -x -k /home/user/.ssh/id_rsa -o list
```

Параметры команды:
-a 10.1.30.249 - IP-адрес сервера, на котором запущен гипервизор KVM;
-l root - логин пользователя для подключения к серверу с гипервизором по ssh;
-n node1 — название виртуальной машины в гипервизоре;
-k /home/user/.ssh/id_rsa - путь к ключу, созданному при помощи команды *ssh-keygen*.
В результате выполнения команда выведет список всех виртуальных машин в гипервизоре.
Теперь следует создать и настроить ресурсы фенсинга для всех узлов кластера.
Выполните следующие действия.
Создайте ресурс фенсинга *fence_node1* для первого узла (*node1*) при помощи команды:

```
sudo pcs stonith create fence_node1 fence_virsh pcmk_host_list="node1" ipaddr="
10.1.30.249" login="root" \
identity_file="/home/u/.ssh/id_rsa" pcmk_reboot_action="reboot"
pcmk_monitor_timeout=60s plug=node1
```

Параметры команды:
pcmk_host_list - какими узлами кластера может управлять данный ресурс;
plug - название виртуальной машины в гипервизоре.
Аналогично создайте ресурс *fence_node2* для узла *node2*.

```
sudo pcs stonith create fence_node2 fence_virsh pcmk_host_list="node2" ipaddr="
10.1.30.249" login="root" \
identity_file="/home/u/.ssh/id_rsa" pcmk_reboot_action="reboot"
pcmk_monitor_timeout=60s plug=node2
```

После создания ресурсов фенсинга для каждого узла, необходимо настроить их таким образом, чтобы они не запускались на тех узлах, для перезагрузки которых они созданы.

Для ресурса *fence_node1* выполните команду:

```
sudo pcs constraint location fence_node1 avoids node1=INFINITY
```

Выполните аналогичную команду для других узлов:

```
sudo pcs constraint location fence_node2 avoids node2=INFINITY
```

Перезагрузите все виртуальные машины в кластере.

Синхронизация ресурсов из файловой системы

Файлы, с которыми работает (модифицирует) Платформа (кроме исполняемых)

1. Файл лицензии `/home/ultima-vmc/Neyross/ultima-vmc/licence`
   Создаётся Платформой при первом запуске и меняется при обновлении лицензии через веб-интерфейс.
   Если лицензирование происходит через HID - файлы лицензии должны быть разные для разных физических машин. Если через несколько Guardant одной поставки - файлы могут быть одинаковы
2. Конфигурационный файл `/usr/share/ultima-vmc/conf/application.conf`
   Создаётся пустым при инсталляции ПО, перезаписывается в процессе первого запуска, в дальнейшем только читается
3. Вспомогательный файл `/home/ultima-vmc/Neyross/ultima-vmc/activiti.cfg.xml`
   Создаётся при первом запуске
4. Если используются локальные ГИС-тайлы, то они лежат в `/home/ultima-vmc/Neyross/ultima-vmc/resources/ultima.vmc.gis.tiles` (загружаются пользователем через веб-интерфейса)
5. Если используется распознавание лиц, то в папке `/home/ultima-vmc/Neyross/ultima-vmc/resources/neurotech` лежат загружаемые через веб-интерфейс ресурсные файлы Нейротека. При этом в папке `/home/ultima-vmc/Neyross/ultima-vmc/resources/neurotech-licence` загружаются лицензионныей файлы Нейротека, **которые должны быть разными для каждой физической машины**
6. ...в будущем в папке `/home/ultima-vmc/Neyross/ultima-vmc/resources/` могут появиться другие директории, которые нужно синхронизировать

Таким образом, должны синхронизироваться файли и каталоги 2, 3, 4 и 6. Файлы и каталоги 1 и 5 должны синхронизироваться в случаях определённых конфигураций и состава.

Для синхронизации объектов файловой системы (файлы и директории) между узлами будем использовать демон *rsyncd* (утилита *rsync*) с соответствующей конфигурвацией *rsyncd.conf*. Проверить налличие утилиты в составе развёрнутой операционной системы можно выполнив запрос состояния сервиса *rsync* (или, в зависимости от версии операционной системы, *rsyncd*):

```
user@node1:~$ sudo systemctl status rsync.service
 rsync.service - fast remote file copy program daemon
    Loaded: loaded (/lib/systemd/system/rsync.service; enabled; vendor preset:
enabled)
    Active: inactive (dead)
  Condition: start condition failed at Tue 2021-11-16 09:35:14 MSK; 5h 12min ago
      Docs: man:rsync(1)
            man:rsyncd.conf(5)

Nov 16 09:35:14 node1 systemd[1]: Condition check resulted in fast remote file copy
program daemon being skipped.
```

> ⚠ **ВНИМАНИЕ**
>
> Для нормальной работы утилиты *rsync* пользователь, от имени которого работает утилита, должен иметь права записи в директорию, хранящую модифицируемые (синхронизируемые) файлы и директории. В нашем случае это будет пользователь *ultima-vmc*.

Таким образом, на каждом из узлов *node1* и *node2* необходимо поменять владельца только для одной директории (остальные находятся в домашней директории пользователя *ultima-vmc*):

```
sudo chown ultima-vmc:ultima-vmc /usr/share/ultima-vmc/conf/
```

Для того, что текущий пользователь имел возможность исполнять удалённо утилиту *rsync* от имени другого пользователя (напомним, в нашем случае от имени пользователя *ultima-vmc*) необходимо предоставить ему эти разрешения и, так как узлы у нас симметричные, то выполнить это необходимо на каждом из узлов *node1* и *node2*. (в примере таким пользователем является пользователь с именем *user*)

```
sudo su
cat > /etc/sudoers.d/user << EOF
user ALL=(ALL) NOPASSWD:/usr/bin/rsync
EOF
exit
```

Если узлы не обменялись ключами сессий *ssh* пользователей как описано в настройках фенсинга Пример 1, то необходимо это выполнить и обменяться ssh ключами между узлами кластера.
Для этого на узле *node1* сгенерируем ключ, передадим его на узел *node2* и проверим сессию

```
ssh-keygen
ssh-copy-id user@node2
ssh user@node2
exit
```

аналогично на узле *node2*

```
ssh-keygen
ssh-copy-id user@node1
ssh user@node1
exit
```

Теперь можно выполнить синхронизацию объектов файловой системы. В нашем случае передать актуальные файлы с узла *node1* на узел *node2*

```
rsync -avz -e ssh --rsync-path="sudo -u ultima-vmc rsync" /usr/share/ultima-vmc/conf
/application.conf user@10.1.30.252:/usr/share/ultima-vmc/conf/application.conf
rsync --rsync-path="sudo -u ultima-vmc rsync" -avz -e ssh /home/ultima-vmc
/Neyross/ultima-vmc/activiti.cfg.xml user@10.1.30.252:/home/ultima-vmc/Neyross
/ultima-vmc/activiti.cfg.xml
rsync -avzr -e ssh --rsync-path="sudo -u ultima-vmc rsync" /home/ultima-vmc
/Neyross/ultima-vmc/resources/ 10.1.30.252:/home/ultima-vmc/Neyross/ultima-vmc
/resources/
```

На этом можно остановиться, т.к. в реальном времени файловые объекты не изменяются в процессе штатной работы прикладных программных средств. Изменения могут возникать при:

- обновлении прикладных программных средств
- расширении функций и/или изменении состава лицензий
- изменении конфигурации прикладных программных средств

Во всех этих случаях, а также при восстановлении узла после сбоя, синхронизацию можно выполнять вручную по завершению восстановительных работ или внесённых изменений.

Автоматическая синхронизация пока не видится целесообразной.

> Автоматическую синхронизацию можно выполнять посредством утилиты *lsyncd*. Данная программа позволяет средствами *rsync* делать резервное копирование сразу же по появлению нового файла в указанной директории. По-сути, выполняется односторонняя синхронизация в реальном времени с помощью Lsyncd.

Lsyncd просматривает дерево локальных директорий с помощью интерфейса модуля мониторинга *inotify*. Он агрегирует и комбинирует события за несколько секунд и затем запускает процесс (или несколько процессов) синхронизации изменений. По умолчанию для этих целей используется *rsync*. Таким образом, *lsyncd* представляет собой легковесное решение для зеркалирования данных, сравнительно легкое в установке, не требующее специфичных файловых систем или блочных устройств, а также не влияющее на производительность файловой системы.

Для установки выполняем команды:

```
sudo apt install lsyncd
```

Разрешаем автозапуск сервиса и изменим пользователя, от имени которого запускается сервис (в нашем случае user):

```
sudo systemctl enable lsyncd
sudo systemctl edit lsyncd
  ...
  User=user
  ...
```

Для настройки и запуска открываем конфигурационный файл:

```
sudo nano /etc/lsyncd.conf
```

Приводим его к виду:

```
settings {
    logfile = "/var/log/lsyncd.log",
    statusFile = "/var/log/lsyncd.stat",
    statusInterval = 5,
    insist = true,
    nodaemon = false,
}
sync {
    default.rsyncssh,
    source = "/usr/share/ultima-vmc/conf/application.conf",
    host = "user@10.1.30.252",
    targetdir = "/usr/share/ultima-vmc/conf/application.conf",
    rsync = {
        _extra = { "-avz --rsync-path='sudo -u ultima-vmc
rsync'" }
    }
}
sync {
    default.rsyncssh,
    source = "/home/ultima-vmc/Neyross/ultima-vmc/activiti.
cfg.xml",
    host = "user@10.1.30.252",
    targetdir = "/home/ultima-vmc/Neyross/ultima-vmc/activiti.
cfg.xml",
    rsync = {
        _extra = { "-avz --rsync-path='sudo -u ultima-vmc
rsync'" }
    }
}
sync {
    default.rsyncssh,
    source = "/home/ultima-vmc/Neyross/ultima-vmc/resources/",
    host = "user@10.1.30.252",
    targetdir = "/home/ultima-vmc/Neyross/ultima-vmc/resources
/",
    rsync = {
        _extra = { "-avzr --rsync-path='sudo -u ultima-vmc
rsync'" }
    }
}
```

где:

settings — общие настройки.

- ***logfile*** — путь до файла логов.
- ***statusFile*** — файл, в который заносятся изменения, найденные с помощью inotify.
- ***statusInterval*** — интервал в секундах для обновления statusFile.
- ***insist*** — позволяет продолжить работу сервиса, даже если одна или несколько целевых директорий недоступны.
- ***nodaemon*** — отключаться или нет от вызывающей стороны. Проще говоря, если разрешить, то будет больше информации по его работе. Для боевого режима можно отключить.

sync — настройка для синхронизации конкретного ресурса. Для каждого создается своя секция sync.

- ***default.rsyncssh*** — в качестве протокола будем использовать rsync через ssh.
- ***source*** — указываем источник данных, откуда синхронизируем данные.
- ***host*** — удаленный компьютер, на который будет идти передача данных. До знака @ указывается пользователь, под которым будет идти подключение.
- ***targetdir*** — каталог на удаленном хосте, в который будет выполняться синхронизация.
- ***rsync, _extra*** — дополнительные ключи запуска rsync. В нашем примере запускаем в режиме архивирования.

После на узле источника (node1) перезапускаем lsyncd:

```
sudo systemctl restart lsyncd
```

Мы можем задать права после синхронизации. Это настраивается в конфигурационном файле */etc/lsyncd.conf* в блоке *sync* раздела *rsync*:

```
sync {
    ...
    rsync = {
        ...
        owner=true,
        chown="ultima-vmc:ultima-vmc"
        chmod="775"
        perms=true
    }
}
```

где:

- ***owner*** *— говорит, сохранять ли владельца файла.*
- ***chown*** *— задает конкретного владельца и группу.*
- ***chmod*** *— задает права на синхронизированные файлы.*
- ***perms*** *— говорит, сохранять ли права.*

При необходимости, мы можем установить некоторые значения для ограничения или обхода ограничений. Настройки задаются в блоке settings:

```
settings {
    ...
    statusInterval = 5
    maxDelays = 900,
    maxProcesses = 6,
}
```

где:

- ***statusInterval*** *— задает интервал обновления статус-файла в секундах. Чем ниже значение, тем быстрее файлы попадают в очередь для синхронизации.*
- ***maxDelays*** *— задает количество файлов в очереди, при достижении которого задачи синхронизации будут запускаться ниже таймера задержки.*
- ***maxProcesses*** *— максимальное количество процессов, которое сможет запустить lsync.*

Мы можем настроить исключение файлов по маске, которые не нужно передавать в другую директорию. Это делается с помощью опций *exclude* или *excludeFrom* в разделе *sync*, например:

```
sync {
    ...
    exclude = { '*.bak' , '*.tmp' },
}
sync {
    ...
    excludeFrom="/etc/lsyncd.exclude",
}
```

в первом блоке мы исключим все файлы, которые заканчиваются на *.bak* или *.tmp*. Для второго мы будем использовать файл */etc/lsyncd.exclude*, в котором перечислим исключения.

Для второго блока создаем файл с исключениями:

```
nano /etc/lsyncd.exclude
```

```
*.tmp
*.bak
testfile.txt
test/
```

в данном примере мы игнорируем файлы, заканчиваются на *.bak* или *.tmp*,а также файл *testfile.txt* и содержимое каталога *test*.

Файл конфигурации сервиса (демона) *rsync* :

```
sudo cat /etc/rsyncd.conf
```

cat /etc/default/rsync

Запуск rsync в режиме демона: sudo rsync --daemon

Создаем ресурс с именем *HA-neyross-rsync* типа *rsyncd* для управления конфигурацией исполняемого сервиса и сразу добавим созданный ресурс в группу, чтобы он запускался вместе на одном узле с другими ресурсами:

```
sudo pcs resource create HA-neyross-rsync lsb:rsync \
   op monitor depth="0" timeout="20s" interval="60s"
   --group master-group
```

Виды планового обслуживания отказоустойчивого кластера

Для проведения регламентных работ необходимо периодически выводить из состава кластера отдельные узлы:

- Выведение из эксплуатации Мастера или Реплики для плановых работ нужно в следующих случаях:
- замена вышедшего из строя оборудования (не приведшего к сбою);
- апгрейд оборудования;
- обновление софта;
- другие случаи.
- Смена ролей Мастера и Реплики. Это нужно в случае, когда, серверы Мастера и Реплики отличаются по ресурсам. Например, у нас в составе отказоустойчивого кластера есть мощный сервер, выполняющий роль Мастера СУБД PostgreSQL, и слабый сервер, выполняющий роль Реплики. После сбоя более мощного сервера Мастера его функции переходят к более слабой Реплике. Логично, что после устранения причин сбоя на бывшем Мастере администратор вернёт роль Мастера обратно на мощный сервер.

> ⚠ **Важно!**
>
> Прежде чем производить смену ролей или вывод Мастера из эксплуатации, необходимо с помощью команды *crm_mon -Afr* убедиться, что в кластере присутствует синхронная реплика. И роль Мастера назначается всегда синхронной реплике.