Servicios de alta disponibilidad.

Autores: Enrique V. Bonet Esteban / Susana Pons Sospedra

Introducción.

La alta disponibilidad es la capacidad de ofrecer un conjunto de servicios sobre un sistema informático que es capaz de recuperarse ante fallos. La alta disponibilidad se relaciona mucho con los sistemas tolerantes a fallos, es decir un sistema que es capaz de funcionar aún cuando se produzca una degradación de los componentes que lo forman.

Es importante no confundir la alta disponibilidad con sistemas de alto rendimiento, pues en este segundo caso lo que se pretende es que el servicio se ofrezca en términos de mejores prestaciones como, por ejemplo, menores tiempos de respuesta o menores tiempos de computo, etc.

De forma general, un cluster de alta disponibilidad esta formado por los siguientes elementos:

- Un conjunto de nodos que proporcionan los servicios del cluster. Cada nodo implementa un conjunto de servicios con una IP única para todo el nodo.
- Un conjunto de estrategias que permitan la migración de los servicios de un nodo a otro tras el fallo del nodo principal. Esto supone reiniciar los servicios siempre y cuando no exista la posibilidad de ejecutar este servicio desde un punto de recuperación (checkpoint).
- Un servicio de comunicación entre los nodos que forman el cluster.
- Un sistema de monitorización para la detección de errores en los nodos del cluster.

Además, y de forma general, todos los cluster poseen un sistema de almacenamiento que permite el acceso a la información almacenada a los nodos que forman el cluster[1].

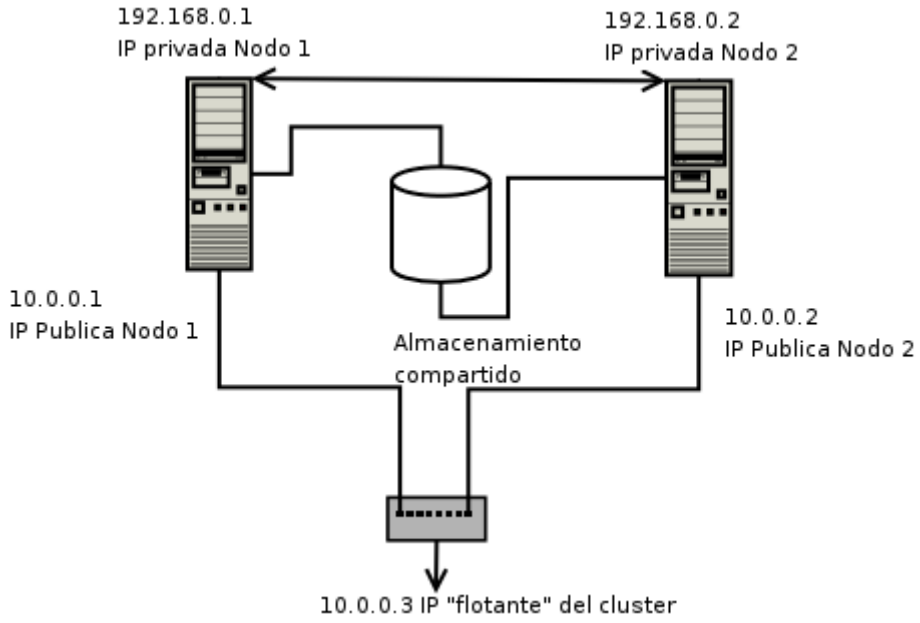
En este tema explicaremos la implementación de un sistema de alta disponibilidad mediante *pcsd*, *corosync* y *pacemaker*[2]. Pcsd proporciona la autenticación de los nodos que forman el cluster y el mecanismo de control del mismo, mientras que corosync proporciona la comunicación entre los nodos del cluster y pacemaker proporciona el control de los recursos en cada uno de los nodos del cluster. Los servicios de *pcsd*, *corosync* y K se arrancan con los comandos:

```
systemctl start pcsd.service
systemctl start corosync.service
systemctl start pacemaker.service
```

[1] Este sistema de almacenamiento puede estar formado por DRBD, tal y como vimos en temas anteriores.

[2] Existen otros paquetes que permiten configurar un sistema de alta disponibilidad, pero corosync/pacemaker es al más utilizado actualmente.

En el ejemplo de configuración que utilizaremos durante todo el tema supondremos dos nodos, de nombres *nodo1* y *nodo2*, de forma que *nodo1* será el nodo principal del cluster y *nodo2* el nodo de respaldo del cluster. Supondremos además que ambos nodos tienen dos interfaces de red, correspondiendo eth0 a una dirección de red privada (192.168.0.1 para *nodo1* y 192.168.0.2 para *nodo2*), que es la que utiliza el almacenamiento compartido mediante DRBD, tal y como vimos en temas anteriores, y eth1 a una dirección de red pública (10.0.0.1 para *nodo1* y 10.0.0.2 para *nodo2*). Por último, supondremos que tenemos una dirección de red 10.0.0.3 que es la dirección pública flotante del cluster, dirección que puede tener asignada uno de los nodos según convenga.



El comando pcs

Actualmente la configuración y manejo del cluster se realiza con el comando *pcs*, el cual posee las siguientes opciones:

Opción	Descripción
resource	Maneja los recursos del cluster
cluster	Maneja y configura las opciones del cluster y de los nodos del cluster.
stonith	Configura la politica de desactivación y reinicio de un nodo cuando el cluster considera que funciona mal.
property	Asigna las propiedades de pacemaker.
constraint	Asigna las restricciones existentes entre los recursos del cluster.
status	Muestra el estado del cluster.
config	Muestra la configuración completa del cluster.

Muchas de estas opciones poseen subopciones que especifican una acción particular. Por ejemplo, *pcs cluster status* informa del estado del cluster, mientras que *pcs cluster force_stop* fuerza la detención de *corosync* y *pacemaker* en el nodo local mediante la señal SIGKILL (valor 9). Veremos las opciones necesarias a medida que vayamos avanzando en el tema.

Configuración de pcsd.

Como hemos comentado, *pcsd* es el paquete que se encarga de configurar la autenticación de los nodos en el cluster y el mecanismo de control del mismo

El primer paso a realizar para su configuración es asignar contraseña a un usuario que será el que ejecutará *pacemaker* y que debe ser común para todos los nodos y con la misma contraseña en todos ellos. De forma general el usuario que se utiliza, si no se desea cambiar la configuración de varios ficheros, es el usuario *hacluster*, usuario que que se ha creado al instalar *corosync* y que será el usuario con el que se ejecutará *pacemaker*. Por tanto, debemos ejecutar el comando:

```
passwd hacluster
```

En todos los nodos del cluster y poner la misma contraseña para todos ellos. Con posterioridad, procedemos a crear la autenticación de los nodos en el cluster mediante el comando:

```
pcs cluster auth nodo1 nodo2
```

Siendo solicitado el usuario del cluster y su contraseña y mostrando el resultado de la operación[3]:

```
Username: hacluster
Password:
nodo1: Authorized
nodo2: Authorized
```

Con este comando, se crean en cada uno de los nodos del cluster, dentro del directorio */var/lib/pcsd* un fichero de identificación del nodo en el cluster, de nombre *pcs_users.conf*, y un certificado de autenticación del nodo, basado en clave pública/privada, que se almacena en los ficheros *pcsd.key* y *pcsd.crt*.

Configuración de corosync.

C*orosync* es el paquete que se encarga de proporcionar la comunicación entre los nodos del cluster. Para ello, utiliza direcciones multicast de forma que todos los nodos que forman un cluster utilizan la misma dirección multicast para comunicarse entre ellos.

[3] Se solicita un único usuario y contraseña, por lo que por este motivo todos los nodos del cluster deben tener configurado el mismo usuario con la misma contraseña.

El fichero corosync.conf.

El fichero de configuración de *corosync* es */etc/corosync/corosync.conf*, que puede crearse de forma manual en cada uno de los nodos del cluster o de forma automática mediante el comando:

```
pcs cluster setup --name miCluster nodo1 nodo2
```

Donde *miCluster* es el nombre que hemos decidido darle al cluster y que, obviamente, puede cambiarse por cualquier otro nombre. La creación de forma automática crea el fichero en uno de los nodos del cluster y propaga automáticamente el mismo al resto de nodos del cluster, por lo que el usuario no debe encargarse de ello.

El fichero *corosync.conf* creado de forma automática o manual está formado por líneas que configuran directivas, las cuales contienen instrucciones sobre el funcionamiento de *corosync* en su interior. Toda línea en blanco o que comienza por el carácter # es ignorada.

Las directivas pueden ser de cinco tipos:

- **totem {}**: Directivas que configuran opciones de comunicación del cluster.
- **logging {}**: Directivas que configuran opciones de log de la aplicación.
- **quorum {}**: Directivas que configuran el número de nodos que se espera formen parte del cluster y su comportamiento.
- **nodelist {}**: Directivas que configuran opciones de los nodos del cluster. Solo puede contener en su interior subdirectivas *node{}* para cada uno de los nodos que son miembros del cluster.
- **qb {}**: Directivas que configuran opciones relacionadas con la librería libqb, que es una librería de subprocesos que proporciona funciones reutilizables para su uso en aplicaciones cliente/servidor de alto rendimiento.

Un ejemplo del fichero *corosync.conf* es el siguiente:

```
totem {
      version: 2
      secauth: off
      clustername: miCluster
      transport: udpu
}

nodelist {
      node {
            ring0_addr: nodo1
            nodeid: 1
      }
      node {
            ring0_addr: nodo2
```

```
            nodeid: 2
      }
}

quorum {
      provider: corosync_votequorum
}

logging {
      to_syslog: yes
}
```

En el ejemplo podemos ver las directivas *totem{}*, *nodelist{}*, *quorum{}* y *logging{}*.

La directiva *totem{}*, que como hemos indicado configura el modo de funcionamiento de la comunicación del cluster, contiene en este caso los valores[4]:

- **version**: Versión del fichero de configuración. El único valor valido es 2.

- **secauth**: Especifica si los mensajes deben ir autenticados o no. El valor por defecto es on (mensajes autenticados), aunque en nuestro caso esta desactivado (valor off).

- **clustername**: Indica el nombre del cluster.

- **transport**: Especifica el protocolo de transporte utilizado. El protocolo por defecto es UDP, aunque puede especificarse UDPU (UDP Unicast) o IBA.

La directiva nodelist*{}* contiene en nuestro caso dos subdirectivas *node{}*, una para cada uno de los miembros del cluster. Estas subdirectivas *node{}* solo contienen dos valores:

- **ring0_addr**: Indica la dirección IP que utilizarán los nodos. En general se pueden definir distintos conjuntos de direcciones IP si se desea configurar distintos clusters modificando el valor ringX_addr, pues distintos valores de X indican distintos anillos (direcciones) a utilizar.

- **nodeid**: Es un valor de 32 bits que especifica el identificador del nodo en el cluster. El valor debe ser distinto para cada nodo del cluster y no puede ser cero, pues es un valor reservado.

Por otra parte, la directiva quorum{} contiene en nuestro caso el valor:

- **provider**: Indica el algoritmo utilizado para comprobar los nodos que actualmente funcionan en el cluster. Actualmente el único valor soportado es *corosync_votequorum*.

Por último, la directiva logging{} tiene el valor:

[4] Una descripción detallada de todas las opciones existentes puede encontrarse en las páginas del manual del sistema.

- **to_syslog**: Indica que el log se envíe al syslog del sistema (valor yes) o no (valor no). El valor por defecto es yes.

Arranque y comprobación del estado de corosync.

Corosync puede arrancarse/pararse de forma manual mediante el comando:

```
pcs cluster {start|stop} [--all]
```

Donde *{start|stop}* indica si se desea arrancar o parar *corosync* y la opción *--all* indica que se ejecute el comando en todos los nodos del cluster, pues en caso contrario solo se ejecuta en el nodo local y es necesario ejecutarlo en todos los nodos del cluster uno a uno.

Una vez arrancado *corosync* en todos los nodos del cluster, podemos comprobar el estado de un nodo del cluster ejecutando el comando:

```
corosync-cfgtool -s
```

Obteniendo como salida, por ejemplo:

```
Printing ring status.
Local node ID 1
RING ID 0
     id = 10.0.0.1
     status = ring 0 active with no faults
```

Además, podemos comprobar todos los valores del cluster mediante el comando:

```
corosync-cmapctl
```

Y si nos interesa solo el estado de los nodos ejecutamos:

```
corosync-cmapctl | grep members
```

Obteniendo como respuesta:

```
runtime.totem.pg.mrp.srp.members.1.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.1.ip (str) = r(0) ip(10.0.0.1)
runtime.totem.pg.mrp.srp.members.1.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.1.status (str) = joined
runtime.totem.pg.mrp.srp.members.2.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.2.ip (str) = r(0) ip(10.0.0.2)
runtime.totem.pg.mrp.srp.members.2.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.2.status (str) = joined
```

Por último, podemos ver el quorum (nodos) que se encuentran actualmente formando el cluster con el comando:

```
pcs status corosync
```

Cuya salida es:

```
Membership information
----------------------
    Nodeid      Votes Name
         1          1 nodo1 (local)
         2          1 nodo2
```

Configuración de pacemaker.

El fichero de configuración de pacemaker es encuentra en */var/lib/pacemaker/cib/cib.xml*, aunque por su complejidad nunca debe editarse directamente, sino que debe utilizarse el comando *pcs* para su configuración y manejo.

Inicialmente, cuando se crea un cluster con *corosync*, los nodos que forman parte del cluster, así como otros parámetros, se configuran por defecto en el citado fichero. Si ejecutamos el comando:

```
pcs config
```

Obtenemos como respuesta:

```
Cluster Name: miCluster
Corosync Nodes:
 nodo1 nodo2
Pacemaker Nodes:
 nodo1 nodo2

Resources:

Location Constraints:
Ordering Constraints:
Colocation Constraints:

Cluster Properties:
 cluster-infrastructure: corosync
 dc-version: 1.1.11-1.fc19-9d39a6b
```

Donde podemos ver como *pacemaker* conoce los dos nodos del cluster. Sin embargo, si chequeamos la corrección de la configuración de *pacemaker* con el comando:

```
pcs cluster verify -V
```

Obtenemos:

```
   error: unpack_resources:     Resource start-up disabled since
no STONITH resources have been defined
   error: unpack_resources:     Either configure some or disable
STONITH with the stonith-enabled option
   error: unpack_resources:     NOTE: Clusters with shared data
need STONITH to ensure data integrity
```

```
Errors found during check: config not valid
```

Los errores que aparecen al chequear la configuración son debidos a la configuración de *stonith.* Stonith es un componente de *pacemaker* que se encarga de reiniciar un nodo cuando se encuentra en un estado indeterminado y que debe ser configurado junto con *pacemaker*. Como *stonith* no esta configurado, pero esta activo, *pacemaker* reconoce este hecho como un error e informa del mismo. Por ello, debemos desactivar *stonith* mediante el comando:

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

Una vez desactivado *stonith*, la configuración inicial de *pacemaker*, es correcta y podemos pasar a su configuración[5]. Como comentario, suele ser recomendable configurar con posterioridad *stonith* y activar el mismo poniendo el valor anterior a true[6].

Una vez desactivado stonith, debemos empezar a configurar algunas propiedades básicas de un cluster[7].

La primera propiedad básica a configurar es el quorum y el comportamiento ante un fallo del quorum. Se dice que un cluster tiene quorum cuando más de la mitad de los nodos que forman el cluster esta online, de forma que pacemaker, si un cluster no tiene quorum detiene todos los servicios del mismo para evitar una posible corrupción de datos. En nuestro caso, como el cluster esta formado por dos nodos y deseamos alta disponibilidad, no podemos permitir este comportamiento de *pacemaker*, por lo que deberemos ejecutar el comando:

```
pcs property set no-quorum-policy=ignore
```

De forma que le indicamos que ignore el hecho de que no exista quorum, por lo que seguirá ofreciendo sus recursos aún con un solo nodo en el cluster, pues deseamos alta disponibilidad y, por tanto, carece de sentido que deje de ofrecer sus servicios por problemas en uno de los nodos.

Otra propiedad que es necesario configurar es el comportamiento de los recursos en los nodos, debemos configurar como se deben repartir los recursos dentro de los nodos cuando se incorporen nuevos nodos al cluster, al haber sido arrancado un nodo del cluster, por ejemplo.

El reparto de los recursos en los nodos se realiza con la propiedad *rsc defaults resource-stickiness*, que puede tomar los valores que se muestran en la siguiente tabla:

[5] Se puede comprobar que la configuración es correcta ejecutando el comando anterior y comprobando como no devuelve ninguna línea de error.

[6] La configuración de *stonith* no se verá en estos apuntes, pues un cluster puede funcionar con *stonith* desactivado, perdiendo unicamente seguridad en el funcionamiento del cluster, pues los nodos no se reinician de forma automática y deben ser reiniciados manualmente si se produce algún error.

[7] Un listado de todas las propiedades que tiene definidas el cluster puede obtenerse ejecutando el comando *pcs property list --all*.

Valor	Descripción
0 (cero)	Los recursos se moverán de un nodo a otro en función de la carga en cada momento.
>0 (mayor que cero)	Los recursos permanecerán en el nodo actual, pero puede moverse a otro nodo cuando se considere necesario. Cuanto más alto es el valor, menor probabilidad de que algún recurso se mueva a otro nodo.
<0 (menor que cero)	Los recursos tienen preferencia a moverse a otro nodo cuando se considere necesario. Cuanto mayor valor absoluto, mayor probabilidad de que algún recurso se mueva a otro nodo.
INFINITY	Los recursos se quedarán en los nodos en los que se encuentran, y solo abandonaran los nodos ante un fallo en los mismos.
-INFINITY	Los recursos se moverán siempre de su ubicación actual.

Por otro lado, la propiedad *symmetric-cluster* de *pacemaker* indica si los recursos del cluster pueden arrancarse en cualquier nodo del mismo (valor true) o debe indicarse explícitamente el nodo donde debe arrancarse (valor false). Como nosotros deseamos configurar un cluster de alta disponibilidad en el que los recursos deben arrancarse en cualquiera de los nodos, configuraremos el cluster como:

```
pcs resource rsc defaults resource-stickiness="INFINITY"
pcs property set symmetric-cluster="true"
```

Si comprobamos ahora la configuración con el comando *pcs config*, obtenemos como salida:

```
Cluster Name: miCluster
Corosync Nodes:
 nodo1 nodo2
Pacemaker Nodes:
 nodo1 nodo2

Resources:

Location Constraints:
Ordering Constraints:
Colocation Constraints:

Cluster Properties:
 cluster-infrastructure: corosync
 dc-version: 1.1.11-1.fc19-9d39a6b
 no-quorum-policy: ignore
 stonith-enabled: false
 symmetric-cluster: true
```

La configuración de los recursos se realiza de acuerdo a la sintaxis:

```
pcs resource create <resource> <[class:provider:]type> [resource
options] [op <operation action> <operation options> [<operation
```

```
action> <operation options>]...] [meta <meta options>...] [--
clone|--master]
```

Donde se puede observar que el único requisito es el nombre que tendrá el recurso <*recurso*> que estamos configurando y el tipo <*type*> del recurso, que corresponde con el nombre del script que se llamará para iniciar y terminar ese recurso. El resto de elementos son opcionales y son necesarios o no según el recurso que estemos configurando, variando además el número de atributos que utilizan.

De los elementos opcionales, la clase *class* especifica la clase del recurso que estamos creando. Podemos obtener las clases existentes mediante el comando:

```
pcs resource standards
```

Que muestra las clases existentes por defecto:

- **ocf**: Open Cluster Framework, que son recursos cuyo script de inicio son una extensión de la convención Linux Standard Base para scripts de inicio, de forma que los valores devueltos, etc., corresponden correctamente con los valores de retorno esperados por *pacemaker* y por tanto son los scripts preferibles para su uso con *pacemaker*.

- **lsb**: Linux Standard Base, que son los scripts de inicio del sistema que corresponden a la especificación SystemV[8], están situados generalmente en */etc/rc.d/init.d*. Pueden dar problemas al no respetar los estándares de *pacemaker* sobre códigos de retorno, etc.

- **systemd**: Son los scripts de inicio del sistema que corresponden a la especificación Systemd. Están situados en el directorio */lib/systemd/system* y pueden presentar los mismos problemas que los scripts *lsb*.

- **service**: Se refiere a cualquier script de inicio del sistema, sea lsb o systemd, por lo que puede considerarse la unión de las dos clases anteriores.

- **stonith**: Usados exclusivamente para la comunicación con recursos relacionados con *stonith*.

Los recursos existentes dentro de cada clase pueden ser listados utilizando el comando:

```
pcs resource agents <class>
```

Por ejemplo si ejecutamos en un sistema el comando:

```
pcs resource agents lsb
```

Una posible respuesta es:

[8] SystemV es el formato de arranque clásico de los sistemas Linux, en que el arranque de los servicios sucede de forma sencuencial, uno detrás de otro. En las últimas versiones de Linux este formato se esta sustituyendo por un arranque en paralelo de todos los servicios que sea posible pues no existan dependencias entre ellos.

```
ceph
drbd
netconsole
network
powerman
```

Que corresponde con los cinco scripts de inicio que posee ese sistema en formato SystemV.

Por otra parte, dentro de una clase, pueden existir varios proveedores distintos que proporcionen recursos para la misma clase. Por ello, el parámetro *provider* permite elegir el proveedor que deseamos utilizar. Los proveedores existentes pueden obtenerse ejecutando el comando:

```
pcs resource providers
```

Y se mostrarán los proveedores de recursos existentes. Por ejemplo:

```
heartbeat
pacemaker
redhat
```

Si de una clase y proveedor en particular deseamos ver los recursos que existen, podemos ejecutar el comando:

```
pcs resource agents ocf:heartbeat
```

Y si se desean obtener todas las clases y proveedores de recursos existentes para cada todas las clases se puede ejecutar el comando:

```
pcs resource list
```

El resto de valores opcionales son utilizados o no según el tipo de recurso que se utilice, poseyendo un número variable de opciones, etc. Si se desea ver todos los valores de configuración, etc., que pueden usarse para un determinado recurso se puede ejecutar el comando:

```
pcs resource describe <clase>:<proveedor>:<tipo>
```

Por ejemplo, si ejecutamos *pcs resource describe ocf:heartbeat:IPaddr2* obtenemos como respuesta:

```
Resource options for: IPaddr2
ip (required): The IPv4 address to be configured in dotted quad notation, for
example"192.168.1.1".
nic: The base network interface on which the IP address will be broughtonline.
...
```

Obteniendo información sobre para que sirve cada uno de los tipos, el uso de los mismos así como el resto de elementos que se necesitan.

Por ejemplo, para configurar la dirección IP flotante del cluster sería necesario ejecutar los comandos:

```
pcs resource create ClusterIP ocf:heartbeat:IPaddr2
ip="10.0.0.3" cidr_netmask=24
pcs resource add_operation ClusterIP monitor interval=30s
```

Donde le estamos indicando que el recurso se llama *ClusterIP*, es de la clase *ocf* y el proveedor es *heartbeat*, siendo del tipo *IPaddr2*. Sus parámetros son la dirección IP flotante a asignar y la máscara de red de esa subred (hemos supuesto una subred /24). Por último, con el segundo comando le estamos indicando que monitorice su estado cada 30 segundos.

De igual forma podemos definir otros recursos, como por ejemplo el uso de un recurso de DRBD para que los dos nodos compartan información. Para ello utilizamos el comando:

```
pcs resource create DrbdDisk ocf:redhat:drbd.sh resource="drbd0"
```

Donde hemos definido un recurso que se llama DrbdDisk, es de la clase *ocf*, proveedor *redhat* y tipo *drbd.sh* y tiene como parámetro que el recurso es *drbd0*, pues suponemos que este es el nombre del recurso compartido[9].

Si a continuación ejecutamos los comando:

```
pcs resource create fileSys ocf:heartbeat:Filesystem
device="/dev/drbd0" directory="/var/lib/mysql" fstype="ext4"
pcs resource add_operation fileSys start interval="0"
timeout="60s"
pcs resource add_operation fileSys stop interval="0"
timeout="60s"
```

Estamos definiendo un recurso llamado *fileSys* que podrá montar el recurso DRBD que hemos definido en el paso anterior. Podemos ver que le hemos pasado el dispositivo a montar (/dev/drbd0), el directorio donde debe montar ese dispositivo (/var/lib/mysql) así como el tipo de fichero (ext4). Además, hemos indicado que tiene 60 segundos para montar y desmontar el sistema de ficheros, dando error en caso de que en ese tiempo no lo haya conseguido.

Podemos ahora indicar que nuestro cluster va a utilizar una base de datos MySQL, y que por tanto debe ejecutar el MySQL mediante el comando:

```
pcs resource create Mysql ocf:heartbeat:mysql
binary="/usr/bin/mysqld_safe"
```

Donde hemos definido un recurso *Mysql* que se arranca mediante el script de *mysql* de la clase *ocf* y proveedor *heartbeat* y que el binario que debe ejecutar es */usr/bin/mysqld_safe*.

[9] Consultar en el tema de servicios de raid en red los nombres de los recursos.

De igual forma, podemos ahora indicar que nuestro cluster va a utilizar otro recurso como un servidor web Apache, y que por tanto debe ejecutar el servicio de httpd mediante los comandos:

```
pcs resource create Httpd ocf:heartbeat:apache params configfile="/etc/httpd/conf/httpd.conf" port="80"
pcs resource add_operation Httpd start interval="0"
pcs resource add_operation Httpd stop interval="0"
pcs resource add_operation Httpd monitor interval="5s" timeout="20s"
```

Donde hemos definido un recurso de nombre *Httpd* que se arranca mediante el script apache de la clase *ocf* y proveedor *heartbeat* con el parámetro *configfile* para indicarle donde se encuentra el fichero de configuración del servidor web y el puerto de escucha 80. Además le hemos indicado que debe arrancar y detener el servicio nada más recibir la orden y por último que debe comprobar que se encuentra activo cada 5 segundos, esperando la respuesta un máximo de 20 segundos.

Como comentario adicional, y dada la importancia del servicio que proporciona el servidor web Apache, indicar que en el fichero */etc/httpd/conf/httpd.conf* del servidor web Apache debería añadirse, en la sección global la línea:

```
PidFile run/httpd.pid
```

Que indica que se cree un fichero *httpd.pid* que contiene el PID del servidor, mientras que al final del fichero, por ejemplo, debería añadirse las líneas:

```
<Location /server-status>
    SetHandler server-status
    <RequireAll>
        Require all granted
        Require ip 127.0.0.1
    </RequireAll>
</Location>
```

Para permitir que *pacemaker* obtenga respuestas a sus comprobaciones de que el servidor Apache se encuentra activo.

Podríamos seguir definiendo recursos, etc., y en cualquier caso se realizaría de forma similar a los ejemplos que hemos visto, por lo que no vale la pena incidir en más en ello,, pudiendo encontrarse información de los parámetros, etc., de cada recurso tal y como se ha comentado con anterioridad.

Una vez hemos definido todos los recursos del cluster existen dos problemas que debemos resolver para que el cluster de alta disponibilidad pueda funcionar de forma correcta.

El primero de estos problemas es indicar que todos los recursos deben ejecutarse en el mismo nodo del cluster, pues para el cluster cada recurso es inicialmente independiente de los otros, por lo que puede decidir asignar la dirección IP a un nodo del cluster, el DRBD a otro nodo del cluster e intentar montar el sistema de ficheros de DRBD en un tercer nodo del cluster (si existiera un tercer nodo). Por tanto, es necesario

indicarle los nodos donde deben arrancar los recursos del cluster. Esto se realiza mediante el comando:

```
pcs constraint colocation add <resource1> <resource2> <score>
```

Donde *<resource1>* y *<resource2>* son dos recursos que hemos definido y *<score>* indica que la relación entre esos dos recursos, de forma que valores positivos de *<score>* indican que los recursos deberían ejecutarse en el mismo nodo y valores negativos indican que los recursos deberían ejecutarse en nodos distintos. Los valores *INFINITY/-INFINITY* cambia la condición deberían por deben, luego obliga a que se ejecuten obligatoriamente en el mismo o en distinto nodo.

En nuestro caso, si ejecutamos los comandos:

```
pcs constraint colocation add ClusterIP DrbdDisk INFINITY
pcs constraint colocation add DrbdDisk fileSys INFINITY
pcs constraint colocacion add fileSys Mysql INFINITY
pcs constraint colocation add Mysql Httpd INFINITY
```

Le obligamos a que todos los recursos, *ClusterIP*, *DrbdDisk*, *fileSys*, *Mysql* y *Httpd* sean ejecutados todos ellos en el mismo nodo del cluster.

El segundo de los problemas es el orden en que se arrancan y terminan los recursos. Si, por ejemplo, el sistema intenta montar el sistema de ficheros de DRBD antes de que el DRBD haya sido puesto en estado primario se producirá obviamente un error, por lo que es necesario indicarle el orden de arranque y parada de los recursos del cluster. Esto se soluciona con el comando:

```
pcs constraint [action] <first rsc> then [action] <then rsc>
[options]
```

Donde action puede tomar los valores *start*, *stop*, *promote* y *demote*, siendo por defecto la acción *start*, y *options* puede tomar los valores *kind={Optional|Mandatory|Serialize}* y *symmetrical={true|false}*, indicando la opción *true* que si se arranca A antes que B, debe detenerse B antes que A.

Así, en nuestro ejemplo deberíamos ejecutar:

```
pcs constraint order ClusterIP then DrbdDisk
pcs constraint order DrbdDisk then fileSys
pcs constraint order fileSys then Mysql
pcs constraint order Mysql then Httpd
```

Indicando que primero debe asignarse la IP del cluster, a continuación poner en estado primario el DRBD, luego montar el sistema de ficheros, arrancar el MySQL y por último ejecutar el servidor web Apache.

Una vez hemos configurado el sistema, si ejecutamos el comando *pcs contraint* obtenemos como salida:

```
Location Constraints:
Ordering Constraints:
```

```
  start ClusterIP then start DrbdDisk
  start DrbdDisk then start fileSys
  start fileSys then start Mysql
  start Mysql then start Httpd
Colocation Constraints:
  ClusterIP with DrbdDisk
  DrbdDisk with fileSys
  fileSys with Mysql
  Mysql with Httpd
```

Donde podemos ver las restricciones que hemos introducido en el sistema. Una vez llegados a este punto, podemos ver la configuración final del cluster volviendo a ejecutar el comando *pcs config*, obteniendo:

```
Cluster Name: miCluster
Corosync Nodes:
 nodo1 nodo2
Pacemaker Nodes:
 nodo1 nodo2

Resources:
 Resource: fileSys (class=ocf provider=heartbeat type=Filesystem)
  Attributes: device=/dev/drbd0 directory=/var/lib/mysql fstype=ext4
   Operations: start interval=0 timeout=60s (fileSys-name-start-interval-0-
timeout-60s)
               stop interval=0 timeout=60s (fileSys-name-stop-interval-0-
timeout-60s)
 Resource: Mysql (class=ocf provider=heartbeat type=mysql)
  Attributes: binary=/usr/bin/mysqld_safe
 Resource: DrbdDisk (class=ocf provider=redhat type=drbd.sh)
  Attributes: resource=drbd0
 Resource: Httpd (class=ocf provider=heartbeat type=apache)
  Attributes: configfile=/etc/httpd/conf/httpd.conf port=80
  Operations: start interval=0 (Httpd-name-start-interval-0)
              stop interval=0 (Httpd-name-stop-interval-0)
              monitor interval=5s timeout=20s (Httpd-name-monitor-interval-5s-
timeout-20s)
 Resource: ClusterIP (class=ocf provider=heartbeat type=IPaddr2)
  Attributes: ip=10.0.0.3 cidr_netmask=24
  Operations: monitor interval=30s (ClusterIP-name-monitor-interval-30s)

Location Constraints:
Ordering Constraints:
    start ClusterIP then start DrbdDisk (Mandatory) (id:order-ClusterIP-
DrbdDisk-mandatory)
   start DrbdDisk then start fileSys (Mandatory) (id:order-DrbdDisk-fileSys-
mandatory)
    start fileSys then start Mysql (Mandatory) (id:order-fileSys-Mysql-
mandatory)
  start Mysql then start Httpd (Mandatory) (id:order-Mysql-Httpd-mandatory)
Colocation Constraints:
    ClusterIP with DrbdDisk (INFINITY) (id:colocation-ClusterIP-DrbdDisk-
INFINITY)
  DrbdDisk with fileSys (INFINITY) (id:colocation-DrbdDisk-fileSys-INFINITY)
  fileSys with Mysql (INFINITY) (id:colocation-fileSys-Mysql-INFINITY)
  Mysql with Httpd (INFINITY) (id:colocation-Mysql-Httpd-INFINITY)

Cluster Properties:
 cluster-infrastructure: corosync
 dc-version: 1.1.11-1.fc19-9d39a6b
 no-quorum-policy: ignore
 stonith-enabled: false
 symmetric-cluster: true
```

Funcionamiento del cluster.

Una vez configurado el cluster, podemos ver el estado del mismo mediante el comando:

```
pcs status
```

Obteniendo como salida:

```
Cluster name: miCluster
Last updated: Thu Apr 10 22:03:20 2014
Last change: Thu Apr 10 22:02:20 2014 via crm_attribute on nodo1
Stack: corosync
Current DC: nodo1 (1) - partition with quorum
Version: 1.1.11-1.fc19-9d39a6b
2 Nodes configured
5 Resources configured


Node nodo1 (1): standby
Node nodo2 (2): standby

Full list of resources:

 fileSys   (ocf::heartbeat:Filesystem):    Stopped
 Mysql     (ocf::heartbeat:mysql):    Stopped
 DrbdDisk  (ocf::redhat:drbd.sh):     Stopped
 Httpd     (ocf::heartbeat:apache):   Stopped
 ClusterIP (ocf::heartbeat:IPaddr2):  Stopped
```

Indicando que tenemos configurados 2 nodos y 5 recursos. Además, los dos nodos se encuentran en estado de standby (parados), por lo que los recursos del cluster no se encuentran disponibles.

Si deseamos arrancar uno de los nodos del cluster basta con ejecutar, en cualquiera de los nodos, el comando:

```
pcs cluster unstandby <nodo>
```

Y el <*nodo*> indicado arrancará. Por ejemplo, si ejecutamos:

```
pcs cluster unstandby nodo1
```

Y comprobamos, ejecutando otra vez el comando *pcs status*, el estado actual del cluster, obtenemos:

```
Cluster name: miCluster
Last updated: Thu Apr 10 22:08:14 2014
Last change: Thu Apr 10 22:08:02 2014 via crm_attribute on nodo1
Stack: corosync
Current DC: nodo1 (1) - partition with quorum
Version: 1.1.11-1.fc19-9d39a6b
2 Nodes configured
```

```
5 Resources configured


Node nodo2 (2): standby
Online: [ nodo1 ]

Full list of resources:

 fileSys   (ocf::heartbeat:Filesystem):    Started nodo1
 Mysql     (ocf::heartbeat:mysql):    Started nodo1
 DrbdDisk  (ocf::redhat:drbd.sh):     Started nodo1
 Httpd     (ocf::heartbeat:apache):   Started nodo1
 ClusterIP (ocf::heartbeat:IPaddr2):  Started nodo1
```

Pudiendo comprobar como *nodo1* ha pasado a estar activo y *nodo2* continua en estado parado, habiendo activado *nodo1* todos los recursos del cluster.

Si ahora ejecutamos, en cualquiera de ellos, el comando *pcs cluster unstandby nodo2* y ejecutamos con posterioridad el comando *pcs status*, obtenemos:

```
Cluster name: miCluster
Last updated: Thu Apr 10 22:10:54 2014
Last change: Thu Apr 10 22:10:52 2014 via crm_attribute on nodo1
Stack: corosync
Current DC: nodo1 (1) - partition with quorum
Version: 1.1.11-1.fc19-9d39a6b
2 Nodes configured
5 Resources configured


Online: [ nodo1 nodo2 ]

Full list of resources:

 fileSys   (ocf::heartbeat:Filesystem):    Started nodo1
 Mysql     (ocf::heartbeat:mysql):    Started nodo1
 DrbdDisk  (ocf::redhat:drbd.sh):     Started nodo1
 Httpd     (ocf::heartbeat:apache):   Started nodo1
 ClusterIP (ocf::heartbeat:IPaddr2):  Started nodo1
```

Pudiendo ver como ambos nodos están en estado activo y por tanto forman parte del cluster, pero como los recursos continua teniéndolos *nodo1*, pues forman un grupo y además hemos indicado que el cluster es simétrico y no deben cambiarse los recursos de un nodo a otro.

Si ahora simulamos un fallo en *nodo1*, lo cual puede hacerse parando ese nodo mediante el comando[10]:

```
pcs cluster standby nodo1
```

[10] Es posible simular un fallo más traumático que el aquí expuesto apagando el equipo de forma súbita, por ejemplo quitando la alimentación, pues un fallo simulado con un rearranque del equipo es similar al que aquí realizamos, pues el sistema al apagarse ejecutara los scripts de parada de corosync y pacemaker y por tanto parará de forma controlada el nodo. De todas formas no resulta recomendable realizar la simulación traumática, pues puede afectar a los sistemas de ficheros, etc., del ordenador.

Y esperamos unos segundos, podemos ver como los recursos son transferidos al *nodo2*, tal y como muestra ahora la salida del comando pcs *status*:

```
Cluster name: miCluster
Last updated: Thu Apr 10 22:13:40 2014
Last change: Thu Apr 10 22:13:25 2014 via crm_attribute on nodo1
Stack: corosync
Current DC: nodo1 (1) - partition with quorum
Version: 1.1.11-1.fc19-9d39a6b
2 Nodes configured
5 Resources configured


Node nodo1 (1): standby
Online: [ nodo2 ]

Full list of resources:

 fileSys   (ocf::heartbeat:Filesystem):    Started nodo2
 Mysql     (ocf::heartbeat:mysql):    Started nodo2
 DrbdDisk  (ocf::redhat:drbd.sh):     Started nodo2
 Httpd     (ocf::heartbeat:apache):   Started nodo2
 ClusterIP (ocf::heartbeat:IPaddr2):  Started nodo2
```

Si ahora volvemos a poner activo *nodo1*, los servicios continuaran siendo proporcionados por *nodo2*, pues hemos indicado en la configuración de *corosync* que ambos nodos son simétricos y por tanto no se transfieren los servicios al otro nodo al volver estar activo.

Como dato final, indicar que si uno de los nodos se encuentra apagado, el comando *pcs status* devolverá como resultado:

```
Cluster name: miCluster
Last updated: Thu Apr 10 22:19:54 2014
Last change: Thu Apr 10 22:15:57 2014 via crm_attribute on nodo1
Stack: corosync
Current DC: nodo1 (1) - partition WITHOUT quorum
Version: 1.1.11-1.fc19-9d39a6b
2 Nodes configured
5 Resources configured


Online: [ nodo1 ]
OFFLINE: [ nodo2 ]

Full list of resources:

 fileSys   (ocf::heartbeat:Filesystem):    Started nodo1
 Mysql     (ocf::heartbeat:mysql):    Started nodo1
 DrbdDisk  (ocf::redhat:drbd.sh):     Started nodo1
 Httpd     (ocf::heartbeat:apache):   Started nodo1
 ClusterIP (ocf::heartbeat:IPaddr2):  Started nodo1
```

Donde puede comprobarse dos datos significativos, el primero es que el estado de ese nodo se indica como *OFFLINE* en lugar de *standby*, y el segundo, que se indica que el estado del cluster es *WITHOUT quorum*, indicando que no existe quorum en el mismo, pero al haber indicado que no es necesario que exista quorum el cluster continua funcionando y ofreciendo sus servicios.