Centro de Procesamiento de Datos

Práctica 3

Configuración de recursos de alta disponibilidad.

Objetivo:

Conocer herramientas (Pacemaker + Corosync) que permiten gestionar recursos para garantizar alta disponibilidad en un cluster de computadores.

Desarrollo:

Partiendo del cluster de la práctica anterior, se van a crear diversos entornos que permitan ofrece un servicio continuo a clientes basado en la redundancia con diversos servidores. Esta práctica está dividida en varios apartados:

1. Instalar corosync
2. Comprobar el funcionamiento básico creando una IP virtuales.
3. Configuración de un servidor web (apache) con alta disponibilidad.
4. Crear un agente personalizado master/slave para gestionar un recurso de alta disponibilidad en Python.

A continuación se describen cada uno de los apartados:

1.Instalar (pacemaker+corosync)

Instalar en los 3 nodos.

yum --enablerepo=base -y install pacemaker libxslt pcs libtool-ltdl cman ccs resource-agents

2.Comprobar el funcionamiento básico creando una IP virtuales.

En esta configuración se van a utilizar los nodos compute-0-0 y compute-0-1 y el nodo frontend como cliente.

Primero activamos la capa de comunicaciones de corosync en compute-0-1 y compute-0-1.
Creamos el fichero de claves con *corosync-keygen.* Este programa tiene una forma especial de obtener los número aleatorios, utilizando /dev/random. Para que funcione correctamente, es preferible ejecutarse enla consola del frontend. Este fichero de claves es el que utilizaremos en el cluster, por lo que lo copiamos con scp a compute-0-0 y compute-0-1.

scp /etc/corosync/authkey compute-0-0:/etc/corosync
scp /etc/corosync/authkey compute-0-1:/etc/corosync

Creamos /etc/corosync/service.d/pcmk

```
service {
        name: pacemaker
        ver:  1
}
```

Partiendo del fichero de configuración que viene como ejemplo

(/etc/corosync/corosync.conf.example), copiar a /etc/corosync/corosync.conf

cambiar en el fichero /etc/corosync/corosync.conf:
bindnetaddr: 10.3.0.0

Copiamos estos ficheros en las máquinas que formarán el anillo, compute-0-0 y compute-0-1

Iniciamos en ambos nodos el servicio corosync y pacemaker:
service corosync start
service pacemaker start
service pcsd start

Verificamos el funcionamiento:
crm_mon -1

Estas siguientes dos órdenes sólo se ejecutan en un servidor ya que se propagan automáticamente al resto.
Desactivamos STONITH
pcs property set stonith-enabled=false
Desactivamos quorum ya que sólo hay 2 nodos:
pcs property set no-quorum-policy=ignore

Creamos usuario hacluster
modificamos password y exportamos al resto de nodos
rocks sync users

Verificamos el fichero /etc/hosts de los nodos compute-0-0 y compute-0-1

Iniciamos la configuracion del cluster:
pcs cluster auth compute-0-0 compute-0-1
pcs cluster setup --name cluster_cpd compute-0-0 compute-0-1
pcs cluster start –all

Verificamos con:
pcs status cluster
pcs status nodes
corosync-cmapctl | grep members
pcs status corosync

Creamos la configuración de la IP virtual:
pcs resource create IPV2 ocf:heartbeat:IPaddr2 ip=10.3.100.100 cidr_netmask=32 op monitor interval="10s”

Comprobamos el funcionamiento
pcs status

Desde el frontend podemos hacer ping a 10.3.100.100

Desactivamos manualmente un nodo para comprobar cómo cambia el recurso:
pcs cluster standby compute-0-0.local

Si hacemos *ssh 10.3.100.100* entramos en el nodo compute-0-1

Reactivamos el nodo compute-0-0:
pcs cluster unstandby compute-0-0.local

El recurso no migra de nuevo porque ya está activo. Podemos cambiarlo manualmente.
pcs resource move IPV2 compute-0-0.local

Activar el envío de una alerta por email cuando cambie el estado:
crm_mon --daemonize –-mail-to <user@example.com> [--mail-host mail.example.com]

Resumen de órdenes pcs:

Mostrar configuración: *pcs cluster cib*
Mostrar estado actual: *pcs status*
Nodo standby: *pcs cluster standby compute-0-0.local*
Activar nodo: *pcs cluster unstandby compute-0-0.local*
Fijar opción: *pcs property set stonith-enabled=false*
Mostrar recursos: *pcs resource standards*
pcs resource agents ocf:pacemaker
Crear un recurso: *pcs resource create IPV2 ....*
Iniciar recurso: *pcs resource start IPV2*
Parar recurso: *pcs resource stop IPV2*
Borrar recurso: *pcs resource delete IPV2*
Actualizar recurso: *pcs resource ipdate IPV2 ...*
Valores por defecto recursos: *pcs resource rsc defauts resoruce-stickiness=100*
Colocation: *pcs constraint colocation add WEB_SITE IPV2 INFINITY*
Orden inicio/parada: *pcs constraint order IPV2 then WEB_SITE*
Localización preferida: *pcs constraint location WEB_SITE prefers compute-0-0.local=50*
Crear clon: *pcs resource clone IPV2 globally-unique=true clone-max=2 clone-node-max=2*
Crear clon M/S: *pcs resource master RES2 RES1 master-max=1 master-node-max=1 clone-max=2 clone-node-max=1 notify=true*

3.Configuración de un servidor web (apache) con alta disponibilidad.

Podemos seguir el ejemplo descrito en:
https://github.com/ClusterLabs/pacemaker/blob/master/doc/Clusters_from_Scratch/en-US/Ch-Apache.txt

Instalamos los paquetes httpd y wget. En resumen, y partiendo de la configuración anterior, se añade el control de un recurso apache:

```
pcs resource create WebSite ocf:heartbeat:apache \
    configfile="/etc/httpd/conf/httpd.conf" statusurl="http://localhost/server-status" \
    op monitor interval="40s"
```

Queremos evitar que un recurso cambie a otro servidor y

pcs constraint colocation add WebSite IPV2 INFINITY

Además, hay qye asignar un orden a la hora que se activen los recursos:

pcs constraint order IPV2 then WebSite

Preferencia de localización:

pcs constraint location WebSite prefers compute-0-0.local=50

Editar el fichero de configuración de apache en ambos nodos: */etc/httpd/conf/httpd.conf*

```
<VirtualHost 127.0.0.1:80>
   ServerAdmin webmaster@dummy-host.example.com
   ServerName localhost
   ErrorLog logs/dummy-host.example.com-error_log
   CustomLog logs/dummy-host.example.com-access_log common
   <Location /server-status>
      SetHandler server-status
      Order deny,allow
      Deny from all
      Allow from 127.0.0.1
   </Location>
</VirtualHost>
```

Comprobemos el funcionamiento

Desactivamos manualmente un nodo para comprobar cómo cambia el recurso: