

► Cluster haute disponibilité avec heartbeat version 2

Le cluster facile avec une interface d'administration graphique ! C'est possible ? Oui, en exploitant les spécificités de la version 2 de heartbeat.

1. Introduction

Dans un article du numéro 97 de *GNU/Linux Magazine* [LM97], nous avons détaillé les étapes et les techniques pour construire un *cluster* (une grappe de machines en français) constitué de 2 machines. Nous avons mis l'accent sur les aspects réseau et sur l'équilibrage de charge apporté par LVS (*Linux Virtual Server*). Nous avons aussi présenté le composant *heartbeat* (voir [HRBT]) que nous avons exploité dans sa version 1, limitée à 2 nœuds par cluster.

Dans cet article, nous allons utiliser les nouvelles fonctionnalités de la version 2 de ce même *heartbeat*, principalement :

- pour construire un cluster de plus de 2 machines ;
- pour surveiller des ressources sans besoin de scripts supplémentaires comme *mon* ou *ldirectord*.

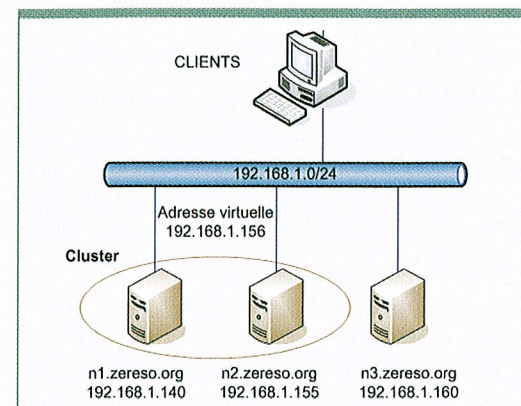
Cette nouvelle mouture est donc plus riche, mais aussi plus complexe à configurer. Elle fait appel à de nouveaux objets, de nouveaux processus, et utilise XML comme langage de description. Aussi, une fois n'est pas coutume, nous allons tout d'abord montrer comment utiliser les outils graphiques livrés avec *heartbeat-2*, puis nous détaillerons les mécanismes internes dans une deuxième étape. La pratique avant la théorie en quelque sorte...

2. Préparation de notre plate-forme de tests

Dans cet article, nous allons mettre en œuvre un cluster minimaliste de 2 nœuds. Nous montrerons comment l'étendre en y ajoutant des nœuds supplémentaires (voir §7). Peu importe la distribution que vous choisirez, pour peu qu'elle respecte les pré-requis suivants :

- Les serveurs connaissent leurs noms et ceux des autres nœuds (cela implique un DNS ou des fichiers */etc/hosts* (!) correctement configurés).
- Les nœuds ont leurs horloges synchronisées (par NTP bien sûr).
- Les nœuds ont été installés avec la même distribution. Cela n'est pas absolument obligatoire, mais cela évite de se creuser la tête quand une ressource ne veut pas fonctionner parce que les chemins d'accès sont différents, parce qu'il manque un *package*, etc.

Dans nos exemples, nous mettrons en œuvre les machines suivantes :



Une des ressources typiques que nous gérons avec le cluster sera une adresse IP virtuelle.

3. Installation

La page de téléchargement est <http://linux-ha.org/download/index.html>. On y trouve les sources, mais aussi des packages pour les principales distributions, à savoir : OpenSuSE, RedHat/CentOS, Fedora, Debian et Gentoo. Cependant, comme nous voulons la dernière version stable, et qu'il n'existe pas de package clés en main pour notre distribution, nous téléchargeons la dernière version stable (à l'heure où nous rédigeons cet article), à savoir *heartbeat-2.1.3.tar.gz*.

L'installation, par package ou bien via les sources, doit se faire sur chacun des nœuds de notre cluster.

3.1 Installation à partir des sources

L'archive *tar.gz* contient tous les composants. L'utilisation habituelle des outils *autoconf* & *make* se fait sans problème.

3.2 Installation à partir des packages

Il faut télécharger les packages pour *heartbeat*, *heartbeat-gui* (l'interface graphique), *pils* et *stonith*. Installez les packages avec votre commande native, *rpm*, *apt-get* ou *emerge*.

pils et *stonith* sont nécessaires pour *heartbeat* et *heartbeat-gui*. *pils* est une bibliothèque qui permet de s'interfacer avec *heartbeat* et de créer des *plugins*. *stonith* contient la mécanique qui permet d'éteindre physiquement un nœud jugé indisponible. Ces deux fonctionnalités ne seront pas développées dans cet article.

heartbeat-ldirectord est uniquement obligatoire pour ceux qui veulent mettre en œuvre LVS (ceux qui ont lu l'article du [LM97] s'en souviennent ?). Nous n'aurons pas besoin de ce package ici.

L'ordre d'installation des packages est donc : *heartbeat-pils*, *heartbeat-stonith*, *heartbeat* et *heartbeat-gui*. N'oubliez pas de reproduire ces opérations sur tous les nœuds du cluster.

3.3 Arborescence des répertoires et fichiers créés

Le tableau suivant liste les principaux répertoires créés lors de l'installation :

Répertoire	Nom logique	Description
/etc/ha.d ou /usr/local/etc/ha.d	HA_DIR	Contient les fichiers de configuration principaux, <i>ha.cf</i> , <i>authkeys</i>
/usr/sbin ou /usr/local/sbin	HA_SBIN_DIR	Contient les exécutables : <i>ha_logger</i> , <i>crm_*</i> , <i>cib*</i>
/usr/local/lib/heartbeat ou /usr/lib/heartbeat	HA_BIN	Scripts complémentaires - des utilitaires
/usr/local/var/lib/heartbeat ou /var/lib/heartbeat	HA_VARLIB	Scripts de description et gestion des ressources gérables par le cluster
/etc/init.d/heartbeat		Script de service
/usr/lib/ocf/	OCF_ROOT	Contient les scripts de gestion des ressources qui suivent la convention OCF (voir §6.2). Avec la configuration « à la heartbeat 1 », ces scripts sont invoqués par les scripts éponymes situés sous <i>\$HA_DIR/resource.d</i> . Avec la version 2, ils sont invoqués directement.

3.4 Comptes et groupes systèmes utilisés

Comme nous n'avons indiqué ni le paramètre *--with-group-name*, ni le paramètre *--with-ccuser-name* au lancement du script *./configure*, les scripts d'installation ont créé un groupe nommé *haclient* et un utilisateur système nommé *hacluster*. L'installation par les packages a créé des noms d'utilisateurs similaires.

Profitez-en pour donner un mot de passe à l'utilisateur *hacluster*. On en aura besoin plus loin !

4. Mon premier cluster !

Commençons simplement avec les machines n1 et n2. L'objectif est de leur faire gérer une adresse IP virtuelle.

4.1 Réalisation d'une configuration de base

Plusieurs fichiers de configuration sont nécessaires à la mise en œuvre de *heartbeat-2*. Pour ceux qui ont déjà utilisé *heartbeat* dans sa version 1, le tableau suivant met en correspondance les anciens et les nouveaux fichiers.

Heartbeat v1	Heartbeat v2	Commentaires
<i>\$HA_DIR/ha.cf</i>	<i>\$HA_DIR/ha.cf</i>	Configuration générale. La v2 supporte plus de paramètres dont celui qui permet d'indiquer qu'on utilise le fichier XML de description des ressources.
<i>\$HA_DIR/authkeys</i>	<i>\$HA_DIR/authkeys</i>	Utilisé pour l'authentification des nœuds. Fichiers identiques dans les 2 versions
<i>\$HA_DIR/haresources</i>	<i>\$HA_RES/crm/cib.xml</i>	Description des ressources
-	<i>\$HA_DIR/logd.cf</i>	Configuration du nouveau démon de gestion des logs

Notons toutefois que *heartbeat-2* peut utiliser une configuration de type *heartbeat-1*. Cependant, dans cet article, nous nous pencherons sur la nouvelle structure des fichiers de configuration.

4.1.1 Fichier *\$HA_DIR/ha.cf*

Nous pouvons nous inspirer du fichier fourni en exemple *doc/ha.cf*. Ce fichier décrit les méthodes de surveillance que doit adopter *heartbeat* pour vérifier l'état de chacun des nœuds du cluster.

Chaque nœud doit surveiller les autres afin de déterminer s'il est opérationnel ou non. Les méthodes de surveillance utilisent des « battements de cœurs » (d'où le nom *heartbeat* !) qui ne sont ni plus, ni moins que des échanges de paquets via un protocole particulier et à travers des liens physiques ou des méthodes multiples.

Un *ping* au sens TCP/IP est une méthode de test, moyennement fiable, mais ce pourrait être une méthode utilisée. Avec 2 nœuds, on peut utiliser un câble série ou réseau croisé pour connecter les deux machines. C'est plus robuste, aucun équipement intermédiaire n'est mis en œuvre. Cependant, nous parlons ici de clusters avec plus de 2 nœuds. Nous utiliserons donc le réseau local Ethernet pour véhiculer les battements de cœur.

Voici un exemple commenté du fichier *ha.cf*. Vous noterez les « nouvelles » directives qui exploitent le nouveau format de description des ressources, ainsi que le nouveau gestionnaire de logs :

```
# Les 3 directives suivantes sont typiques de la version 1,
# donc elles sont en commentaire...
#logfile local7
#logfile /var/log/ha-log
#debugfile /var/log/ha-debug

# Maintenant, on utilise un nouveau démon pour
# la gestion des logs :
use_logd yes

# Description des mécanismes de "battements de coeur":
# paquets UDP sur le port 694
udpport 694

# délai entre 2 battements (en secondes)
keepalive 2
# temps nécessaire avant de déclarer un nœud comme mort
deadtime 10
# valeur utilisée pour le démarrage (> 2 fois le deadtime)
initdead 80
```

```
# quand le noeud qui possède une ressource tombe, on ne
# souhaite pas qu'il recouvre cette ressource automatiquement
auto_failback off

# medium de transmission :
# (je préfère le Multicast au Broadcast)
# bcast eth0
# attention: il y a une erreur volontaire ci-dessous: c'est exprès !
multicast eth0 255.0.0.7 694 1 0

# Liste de noeuds
# le nom indiqué DOIT être celui retourné par 'uname -n' et
# chaque noeud DOIT pouvoir résoudre ces noms !
node n1.zereso.org
node n2.zereso.org

# Ah ! choix du nouveau format des ressources :
crm yes
```

4.1.2 Fichier \$HA_DIR/authkeys

Ce fichier décrit les méthodes d'authentification utilisées par les nœuds d'un même cluster pour communiquer. Copiez le fichier `doc/authkeys` sous `$HA_DIR/` et choisissez un mot de passe commun ainsi qu'une méthode de chiffrement, par exemple :

```
auth 1
1 sha1 "super-secret"
```

Les permissions sur ce fichier doivent être 600 :

```
# chmod 0600 $HA_DIR/authkeys
```

4.1.3 Fichier \$HA_DIR/logd.cf

Dorénavant, les messages de logs seront envoyés au démon `ha_logd` situé dans le répertoire `HA_BIN`. Ce démon est lancé automatiquement par le script de démarrage `heartbeat`. L'intérêt de ce nouveau démon est justifié par sa capacité à envoyer les logs vers plusieurs destinations (fichiers, `syslog`) tout en étant non bloquant pour les processus appelants.

Voici un exemple de configuration :

```
# ha_logs peut écrire dans ses propres fichiers
debugfile /var/log/ha-debug
logfile /var/log/ha-log

# il peut aussi communiquer les logs à syslog
logfacility daemon

# Marqueur en tête des messages envoyés à syslog
entity logd

# enregistrement auprès de apphbd (démon qui permet de surveiller des application.
# Ici on pourrait faire surveiller heartbeat par apphbd)
useapphbd no

# ha_logd permet de définir des tailles de queues de messages
# taille pour la queue utilisée par le processus qui lit les messages
# provenant des clients et qui les dispatch vers les processus fils
sendqlen 256
# taille de la queue utilisée par les fils pour recevoir les messages
# émis par le processus parent. Ce sont les fils qui écrivent vraiment
# sur le disque ou émettent le message à syslog
recvqlen 256
```

4.1.4 Fichier \$HA_DIR/haresources

Eh non ! Il n'existe plus avec `heartbeat-2` si l'on a indiqué l'option `crm yes` dans le fichier `ha.cf`. `heartbeat-2` met en œuvre un véritable *Cluster Resource manager* (CRM) qui utilise des fichiers XML.

Pour l'instant, poursuivons sans créer de ressources et vérifions que notre cluster est fonctionnel.

4.2 Démarrage du cluster

Nous pouvons alors démarrer `heartbeat` sur n1 par la commande suivante :

```
n1 # /etc/init.d/heartbeat start
Starting High-Availability services:
2008/01/03_16:41:52 INFO: Resource is stopped
heartbeat[16470]: 2008/01/03_16:41:52 ERROR: Illegal directive
[multicast] in /usr/local/etc/ha.d/ha.cf
heartbeat[16470]: 2008/01/03_16:41:52 ERROR: Heartbeat not started:
configuration error.
heartbeat[16470]: 2008/01/03_16:41:52 ERROR: Configuration error,
heartbeat not started. [ÉCHOUÉ]
```

Le démarrage a échoué. L'erreur apparaît clairement à l'écran, sinon il aurait fallu consulter le log des erreurs (`/var/log/messages`). Éditez le fichier `ha.cf` et changez le nom de la directive `multicast` en `mcast`. Relancez le script, aucune erreur ne devrait s'afficher et vous pouvez vérifier que les processus `ha_logd` et `heartbeat` sont actifs.

```
n1 # /etc/init.d/heartbeat start
Starting High-Availability services: [OK]
n1 # pidof heartbeat
4115 4114 4113 4110
n1 # pidof ha_logd
4089 4088
```

Vous pouvez alors recopier les fichiers `$HA_DIR/{ha.cf,authkeys,logd.cf}` sur n2, puis lancer `heartbeat` sur ce nœud.

N'oubliez pas que, pour l'instant, dès que vous ferez une modification dans la configuration d'un nœud, il faudra recopier cette modification sur tous les nœuds du cluster et relancer `heartbeat` en conséquence.

4.3 Tests de bon fonctionnement

Vérifions le statut de `heartbeat` sur la machine locale :

```
n1 # cl_status hbstatus
Heartbeat is running on this machine.
```

Nous pouvons aussi obtenir la liste des nœuds actifs :

```
n1 # cl_status listnodes
n1.zereso.org
n2.zereso.org
```

Comme nous utilisons le nouveau gestionnaire de ressource, le CRM (*Cluster Resource Manager*), nous disposons de nouvelles commandes ! Par exemple, `crm_mon` qui affiche en mode texte l'état du cluster. La commande suivante affiche toutes les 5 secondes l'état de tous les nœuds :

```
# crm_mon -i5
=====
Last updated: Thu Jan 3 19:47:36 2008
Current DC: n2.zereso.org (f3643212-9c27-4f11-88b9-83f5eb29bd39)
2 Nodes configured.
0 Resources configured.
=====
Node: n2.zereso.org (f3643212-9c27-4f11-88b9-83f5eb29bd39): online
Node: n1.zereso.org (a328e624-2655-47d5-8c38-5f7ccb2c90f0): online
```

Mission accomplie ! Les 2 nœuds sont actifs (*online*). La valeur entre parenthèses, vous l'aurez deviné, est un UUID, valeur unique associée à chaque nœud.

Si vous souhaitez surveiller votre cluster avec `nagios`, vous pouvez utiliser la commande suivante :

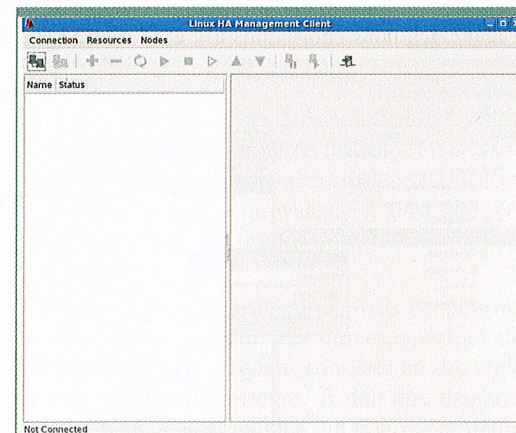
```
# crm_mon -s
Ok: 2 nodes online, 0 resources configured
```

4.4 Ajout d'une première ressource

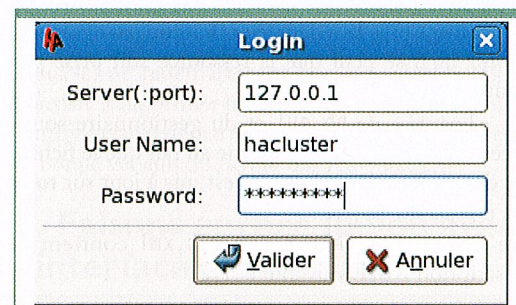
Ajoutons des ressources au cluster. Notre premier exemple sera une adresse IP virtuelle.

Nous allons utiliser `hb_gui`, une interface réalisée en Python-Gtk. Cette interface peut être lancée sur n'importe quelle machine disposant d'un serveur X. La lancer sur un nœud du cluster n'est donc pas obligatoire.

```
n1 # hb_gui &
```



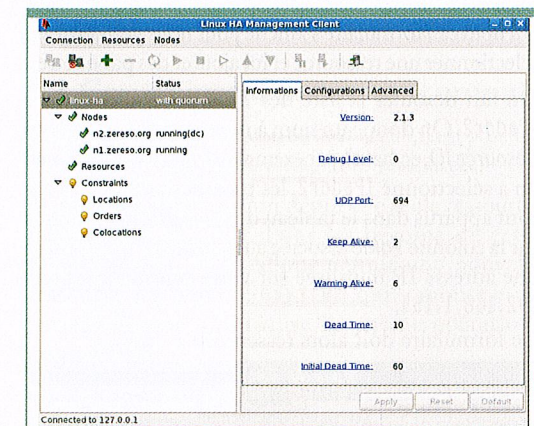
La 1ère action à effectuer consiste à se connecter au cluster en choisissant l'option `Connection/Login...` du menu. Quand vous cliquez sur l'icône de connexion, une boîte de dialogue apparaît :



Indiquez le nom ou l'adresse IP de n'importe quel nœud du cluster. Le nom d'utilisateur est un nom connu par le système hôte, mais il doit appartenir au

groupe `haclient` (voir §3.4) (`root` n'est probablement pas un membre de ce groupe !). Si vous utilisez le compte `hacluster`, vous avez dû fixer son mot de passe à l'étape §3.4.

Si les informations indiquées sont correctes et que `heartbeat` fonctionne sur la machine indiquée (ici, c'est la machine locale), une connexion s'établit en SSL, et l'état du cluster et de ses ressources apparaît alors, après quelques secondes d'attente :



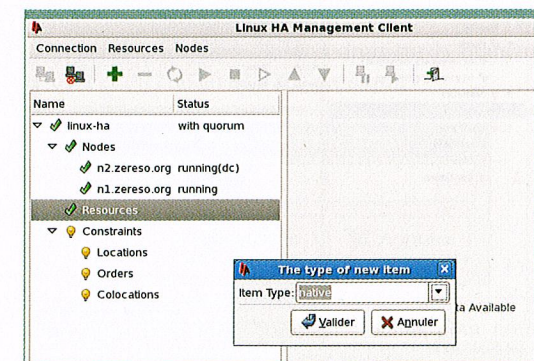
ATTENTION

Attendez une bonne dizaine de secondes après le lancement du nœud avant de vous y connecter...

L'outil permet de modifier certains paramètres du cluster, mais pas tous (il faut donc suivre l'évolution des versions ou bien proposer des patches !). À gauche, l'arbre présente les ressources et les nœuds, à droite, on obtient les paramètres de l'objet sélectionné dans l'arbre.

Si l'on clique sur la branche `Nodes`, on obtient la liste des nœuds. Ici, `n2.zereso.org` est désigné comme le DC (*Designated Coordinator*) : c'est lui qui gère le cluster et dit aux autres nœuds quoi faire. N'ayez pas peur ! Le DC n'est pas un SPOF (*Single-Point of Failure*). S'il devient inopérant, son rôle sera pris par un autre nœud.

N'oublions pas notre objectif : nous devons créer une ressource. Pour cela, sélectionnez la branche `Resources` dans l'arbre, puis cliquez sur le bouton `+` de la *toolbar*... La boîte de dialogue suivante nous demande de choisir un type de ressource :



Les types peuvent être :

- **native** : c'est ce que nous choisissons ici (correspond aux scripts dits « de service », ainsi qu'aux ressources gérées par **heartbeat**) ;
- **group** : permet de définir des groupes de ressources qui doivent toujours être associées ;
- **location**, **order** et **colocation** : définissent des contraintes d'allocation des ressources que nous présenterons au chapitre §5.

Un nouveau formulaire s'affiche pour nous permettre de sélectionner une ressource et lui ajouter des paramètres.

On fait dérouler la liste des **Type** pour sélectionner **IPaddr2**. On donne un nom à notre ressource (champ **Resource ID**, en haut), par exemple : **ipaddr_test**. Quand on a sélectionné **IPaddr2**, les paramètres obligatoires sont apparus dans le tableau des paramètres. Cliquez sur la colonne **Value** associée au paramètre **ip**. Saisissez une adresse IP inutilisée sur votre réseau local ; ici : **192.168.1.155**.

Le formulaire doit alors ressembler à ceci :

Puis, validons notre ressource en cliquant sur le bouton **Ajouter**.

Si notre ressource est correctement définie, elle apparaît dans l'arbre sous la branche **Resources**. Un « x » rouge indique que quelque chose ne fonctionne pas. Une marque verte indique un fonctionnement normal. Dans notre cas, la ressource n'est pas activée. Il faut la sélectionner dans l'arbre, puis cliquer sur le bouton **Play** de la toolbar.

Le paramètre **target_role** passe alors de **not running** à **started**. Dans l'arbre, on peut constater qu'un nœud a « pris » l'adresse IP virtuelle, ici c'est **n2** :

Un ping sur l'adresse IP virtuelle (ici, **192.168.1.155**) permet de vérifier rapidement que la ressource de type **IPaddr2** est bien active.

L'interface permet aussi de rajouter des paramètres à la ressource. Dans notre cas, nous pouvons spécifier sur quelle interface l'adresse IP virtuelle doit être activée. Cliquez sur l'onglet **Parameters**, à droite, puis sur le bouton **Add Parameter**, une boîte de dialogue s'affiche qui permet de sélectionner un paramètre supporté par **IPaddr2**. Ici, nous choisissons **nic** et la valeur **eth0**.

Validez le nouveau paramètre, puis appuyez sur le bouton **Apply**. Pour que ce paramètre soit pris en compte, il faut stopper la ressource (bouton carré de la Toolbar), puis la redémarrer (bouton triangulaire). Notez qu'il se peut que la ressource soit prise par l'autre nœud...

La « beauté » de **hb_gui** et du gestionnaire sous-jacent (le CRM) tient en partie au fait que le fichier de description des ressources est mis à jour sur **tous** les nœuds du cluster !

Ce fichier, **\$HA_VARLIB/crm/cib.xml** contient la description XML suivante :

```
<cib admin_epoch="0" have_quorum="true" num_peers="2" cib_feature_revision="1.3"
  ignore_dtd="false" ccm_transition="2" generated="true"
  dc_uuid="f3643212-9c27-4f11-88b9-83f5eb29bd39" epoch="6" num_updates="53"
  cib-last-written="Fri Jan 4 12:37:03 2008">
```

```
<configuration>
  <crm_config>
    <cluster_property_set id="cib-bootstrap-options">
      <attributes>
        <nvpair id="cib-bootstrap-options-dc-version" name="dc-version"
          value="2.1.3-node: 552305612591183b1628baa5bc6e903e0f1e26a3"/>
      </attributes>
    </cluster_property_set>
  </crm_config>
  <nodes>
    <node id="f3643212-9c27-4f11-88b9-83f5eb29bd39" uname="n2.zereso.org" type="normal"/>
    <node id="a328e624-2655-47d5-8c38-5f7ccb2c90f0" uname="n1.zereso.org" type="normal"/>
  </nodes>
  <resources>
    <primitive class="ocf" type="IPaddr2" provider="heartbeat" id="ipaddr_test">
      <meta_attributes id="ipaddr_test_meta_attrs">
        <attributes>
          <nvpair name="target_role" id="ipaddr_test_metaattr_target_role"
            value="started"/>
        </attributes>
      </meta_attributes>
      <instance_attributes id="ipaddr_test_instance_attrs">
        <attributes>
          <nvpair id="fc323ebd-723e-4111-a20d-4f62e65f04eb"
            name="ip" value="192.168.1.155"/>
          <nvpair id="930da5e5-26ed-46f2-8014-ba034d50e850" name="nic"
            value="eth0"/>
        </attributes>
      </instance_attributes>
    </primitive>
  </resources>
  <constraints/>
</configuration>
```

On y retrouve la liste des nœuds du cluster (balises **<nodes>** et **<node>**), les ressources (balises **<resources>** et **<primitive>**) avec leurs paramètres (balise **<attributes>** et **<nvpair>** – paire nom/valeur –), ainsi que l'état du cluster, l'état des nœuds et la localisation des ressources.

Notre exemple est simplissime, mais on pourrait, en définissant plusieurs ressources, spécifier des **contraintes**, qui pourraient consister en des règles de dépendances (la ressource A doit être démarrée après la ressource B) ou bien qui pourraient définir des règles d'allocation (la ressource Apache ne doit fonctionner que sur les nœuds n2, n4 ou n5, etc.).

Quand une modification est effectuée sur le fichier des ressources, la nouvelle version est propagée à partir du DC vers les autres nœuds. Ne modifiez jamais le fichier **cib.xml** « à la main » sur les autres nœuds. Ceux-ci ne redémarreront pas. La solution, alors, consiste à supprimer le fichier **cib.xml** sur les nœuds fautifs, qu'on redémarre ensuite. Le DC leur renverra la bonne version du fichier des ressources.

5. Faisons encore joujou avec l'interface graphique

Dans ce chapitre, nous allons explorer d'autres fonctionnalités de **heartbeat-2**, en particulier, les contraintes qui régissent l'ordre de démarrage et la localisation des ressources.

Les contraintes peuvent être obligatoires (**mandatory**) et imposer que la ressource soit activée sur un sous-ensemble précis des nœuds du cluster ou bien, elles peuvent être préférentielles et les ressources sont alors activées sur le nœud dont le poids (le « score ») est le plus élevé.

Par défaut, le cluster est symétrique (**symmetric_cluster = true**), donc, en l'absence de contraintes explicites, les ressources peuvent s'exécuter sur tous les nœuds.

5.1 Contrainte d'ordonnement

Dans cette nouvelle version de **heartbeat**, il est possible de définir l'ordre de démarrage des ressources.

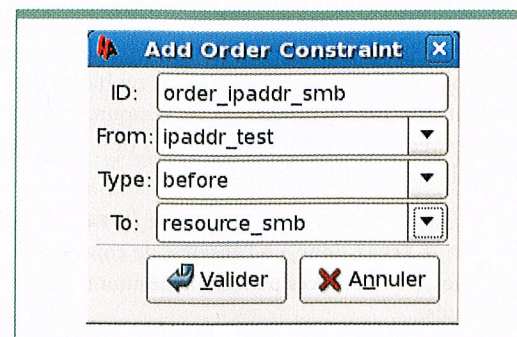
Nous avons pour l'instant défini une seule ressource, instance de **IPaddr2**. Pour les besoins de cette démonstration, nous allons définir une nouvelle ressource de type **Filesystem** qui « monte » un partage Samba. On réalise les mêmes opérations qu'au §4.4.

On clique avec le bouton droit sur la branche **Resources** de l'arbre, puis on choisit **Add New Item**. On sélectionne une ressource de type **native**. On remplit alors la boîte de dialogue qui apparaît, ainsi :

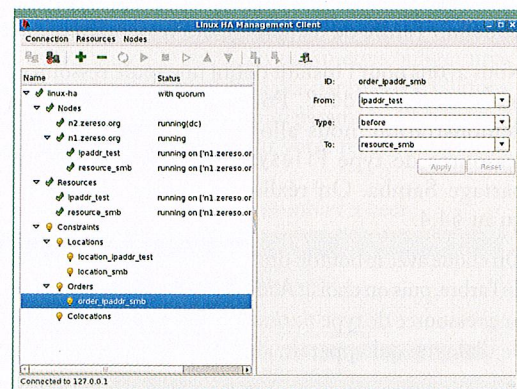
Nous donnons dès maintenant les valeurs des paramètres obligatoires (**device**, **directory** et **fstype**). Nous avons aussi ajouté le paramètre facultatif **options**.

Nous souhaitons maintenant (c'est juste un exemple, cela n'est pas obligatoire en réalité, ici), que la ressource **ipaddr_test** soit démarrée avant **resource_smb**. Pour cela, on clique avec le bouton droit sur la branche **Orders** et on choisit **Add a New item** :

On valide. Une autre boîte est affichée pour nous permettre de caractériser la contrainte :



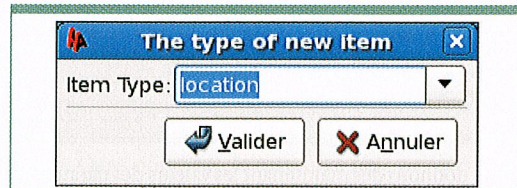
On valide cette ultime boîte de dialogue. La contrainte est activée :



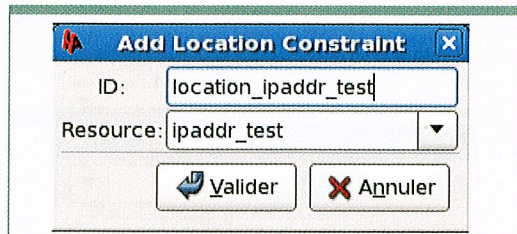
5.2 Contrainte de localisation

Il est parfois intéressant de pouvoir forcer la migration d'une ressource sur nœud ou un ensemble de nœuds du cluster. Pour cela, on peut définir des contraintes de « relocation ».

À partir de l'interface graphique **hb_gui**, on clique avec le bouton droit sur la branche **Locations**, et on choisit **Add a New Item** :

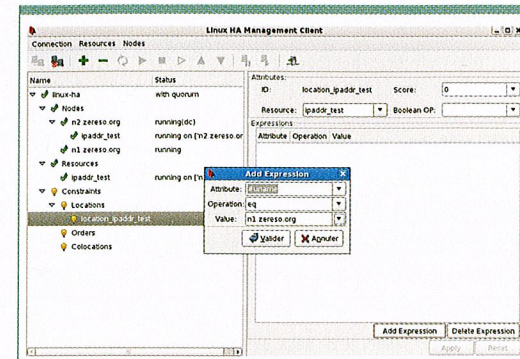


On valide. Une autre boîte est affichée pour nous permettre de donner un nom à la contrainte et de choisir la ressource contrainte :



On valide. La contrainte apparaît dans l'arbre de gauche et, à droite, on visualise les attributs de cette contrainte. En cliquant sur le bouton **Add Expression**, on ajoute une expression. Ici, nous décidons que la

ressource doit être placée en priorité sur le nœud dont le nom (**#uname**) est égal (**eq**) à **n1.zereso.org**.



On valide la contrainte en appuyant sur **Apply**, et là... on constate que la ressource n'est pas forcément relocalisée. En fait, toutes les contraintes ont un score qui influe sur la localisation de la ressource. Par défaut, la valeur du score est 0. Dans la **combo-box** en haut à droite, on choisit alors la valeur prédéfinie **+INFINITY**, on clique encore sur **Apply** et, là, la ressource est migrée immédiatement. C'est normal, son score est devenu maximal.

5.3 Contrainte de regroupement

Il est aussi possible d'indiquer qu'une ressource doit ou ne doit pas être activée sur le même nœud que telle autre ressource. On parlera alors de « co-location ».

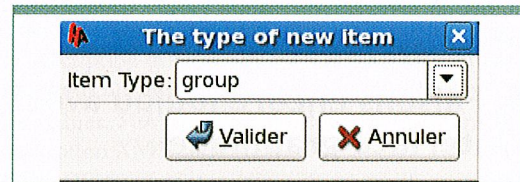
Nous laissons le soin au lecteur de définir une telle contrainte à l'aide de l'interface graphique.

5.4 Groupes de ressources

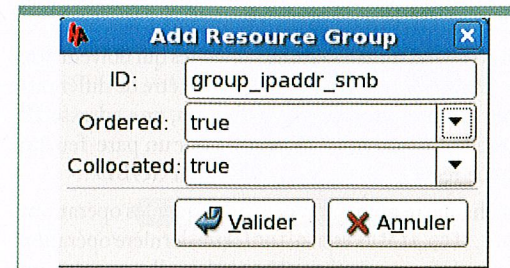
Pour simplifier la gestion de ressources interdépendantes, il est possible de définir des groupes de ressources. À l'intérieur d'un même groupe, les règles par défaut suivantes s'appliquent :

- ▶ Les ressources sont activées séquentiellement selon leur ordre de déclaration dans le groupe (**start-after** implicite).
- ▶ Les ressources doivent être activées sur le même nœud (co-location implicite)

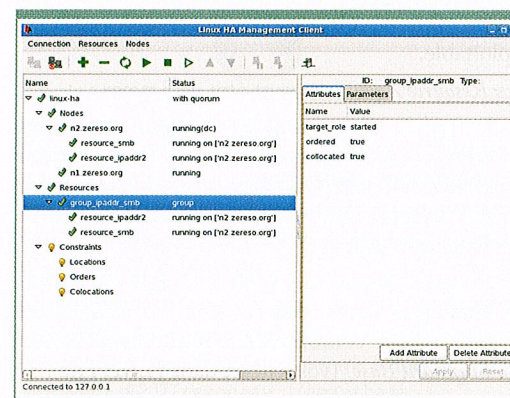
Avec notre maquette, plutôt que de définir une contrainte d'ordonnancement et une contrainte de regroupement, nous pouvons définir un groupe. Pour cela, nous détruisons les contraintes définies et nous créons un groupe en créant une nouvelle ressource de type « **Group** » :



Nous donnons ensuite un ID au groupe et nous laissons la valeur par défaut des attributs d'ordonnancement et de co-localisation :



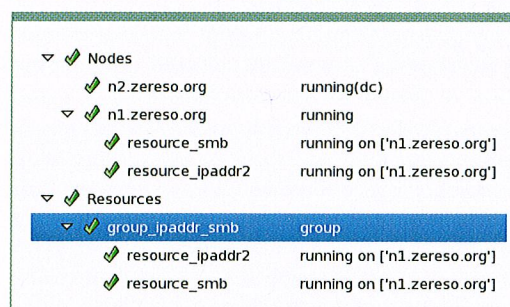
La boîte suivante nous incite à définir au moins une ressource native qui sera associée au groupe (voir §4.4). Ici, nous redéfinissons une ressource de type **IPaddr2**. Dans l'arbre à gauche, nous voyons apparaître la ressource et son contenu. Nous ajoutons aussi un nouveau groupe, l'équivalent de l'ancienne ressource **Filesystem** (**resource_smb**) définie au §5.1.



Pour activer les ressources du groupe, il faut sélectionner celui-ci, puis appuyer sur le triangle vert. Attention, il se peut qu'un avertissement soit affiché. Cela concerne la précedence des ordres émis par le « groupe » sur les attributs placés dans les ressources du groupe, en particulier pour l'attribut **target_role**. Validez la boîte en demandant d'effacer les valeurs de ces attributs.

On peut tester l'effet « groupe » en demandant la migration des ressources d'un nœud vers l'autre. Ici, nous allons migrer le groupe de **n2** vers **n1**. L'interface graphique ne permet pas de réaliser cette opération. Nous utilisons pour cela les commandes en ligne :

```
# crm_resource -M -r group_ipaddr_smb -H n1.zereso.org
```



La migration a réussi : du nœud **n2** (voir écran plus haut), les ressources sont passées à **n1**.

Le chapitre §7 détaillera un certain nombre d'opérations réalisables avec les commandes en ligne.

5.5 Contraintes temporelles

Les règles peuvent contenir des contraintes temporelles qui vont conditionner les valeurs des attributs ou les valeurs des paramètres du cluster. L'interface graphique ne gère pas (encore) ce type de contraintes : il faut donc modifier le fichier **cib.xml** manuellement.

Un exemple typique serait d'autoriser le **failback** automatique uniquement pendant le week-end. Pour définir cette règle, on utilise le paramètre **default_resource_stickiness** du cluster. Plus sa valeur est élevée, plus la ressource a de « chances » de rester sur le même nœud. L'exemple ci-dessous est extrait de la présentation d'Alan Robertson [TUTO] : nous y avons ajouté nos propres commentaires :

```
<!-- les "id" sont informatifs et doivent être uniques -->
<crm_config>
  <!-- les samedi et dimanche (6-7), la valeur du paramètre
        "default_resource_stickiness" est égale à 0, donc la ressource peut être
        déplacée d'un nœud actif vers un autre nœud actif -->
  <cluster_property_set id="weekend_override" score="100">
    <rule id="my_ip:failover" boolean_op="and">
      <date_expression id="my_ip:days" operation="date_spec">
        <date_spec id="my_ip:days" weekdays="6-7"/>
      </date_expression>
    </rule>
  </cluster_property_set>
  <attributes>
    <nvpair id="sat-sun-stick" name="default_resource_stickiness"
      value="0"/>
  </attributes>
</crm_config>

<!-- en fait, la propriété suivante n'ayant pas de règle (rule),
      c'est la règle par défaut. Elle donne la valeur maximale (INFINITY)
      au paramètre "default_resource_stickiness", donc, une fois qu'un nœud
      prend une ressource, il ne la rend pas si le nœud défaillant revient en jeu -->
<cluster_property_set id="default_cluster_properties" score="10">
  <attributes>
    <nvpair id="default-sticky" name="default_resource_stickiness"
      value="INFINITY"/>
  </attributes>
</cluster_property_set>
...
</crm_config>
```

6. Un peu de mécanique maintenant !

Dans ce paragraphe, nous allons parler un (peu) de l'architecture de **heartbeat**, puis nous allons voir comment écrire notre propre script de gestion de ressources, en respectant la norme OCF.

6.1 Liste des différents composants

Le tableau suivant donne la liste des processus qui composent le système **heartbeat**. Certains processus sont actifs sur tous les nœuds, d'autres uniquement sur le DC (le chef !).

Nom logique	Processus	Rôle
Cluster Resource Manager Daemon	crmd	Coordonne les échanges entre les différents composants du <i>Cluster Resource Manager</i> local, c'est-à-dire : <i>heartbeat</i> , <i>lrmd</i> , <i>ccm</i> , <i>ha_logd</i> , <i>stonithd</i> (voir ci-dessous)
Designated Coordinator (DC)	crmd	Instance particulière d'un <i>Cluster Resource Manager Daemon</i> . Le nœud auquel échoit ce rôle est élu par les nœuds actifs du cluster. Son rôle consiste à allouer les ressources sur les nœuds. C'est sur le DC que sont effectuées les modifications de la <i>Cluster Information Base</i> (CIB) (pensez au fichier <i>\$HA_VARLIB/crm/cib.xml</i> ci-dessus). Les autres <i>Cluster Resource Manager Daemon</i> sur les autres nœuds sont alors des esclaves qui reçoivent une copie de la CIB.
Cluster Consensus Manager	ccm	S'assure en temps-réel que les nœuds d'un même cluster peuvent dialoguer entre eux. Détermine qui est dans le cluster et qui n'y est pas.
Resource Manager	-	Ensemble des composants suivants : <i>Cluster Information Base</i> <i>Cluster Resource Manager Daemon</i> <i>Policy Engine</i> <i>Transition Engine</i> <i>Local Resource Manager</i>
Policy Engine (PE)	pengine	Processus fils de <i>crmd</i> . Ne fonctionne que sur le DC. Contrôle l'emplacement des ressources dans le cluster en fonction des contraintes édictées dans la CIB
Transition Engine (TE)	tengine	Processus fils de <i>crmd</i> . Ne fonctionne que sur le DC. Demande aux démons <i>lrmd</i> distants (voir ci-dessous) d'effectuer les actions correspondantes au graphe de transition calculé par le PE
Local Resource Manager	lrmd	Réalise les opérations sur les ressources. Il utilise des scripts appelés <i>Resource Agent</i> pour démarrer, stopper, surveiller, donner l'état ou lister les ressources.
Cluster Node	-	Hôte sur lequel sont activés <i>heartbeat</i> , le <i>Cluster Consensus Manager</i> et le <i>Cluster Resource Manager</i>
Stonith Daemon	stonithd	Met en œuvre le <i>fencing</i> : l'art d'éradiquer un nœud mort ou malade
Logging Daemon	ha_logd	Sur chaque nœud, ce démon est chargé du <i>dispatching</i> des logs
Heartbeat	heartbeat	Processus de gestion de la communication inter-nœuds.
Attribute Daemon	attrd	Permet de modifier dynamiquement les valeurs des attributs utilisés pour les contraintes dans la CIB

6.2 Définition des ressources

Les *Resources* sont des entités concrètes qui doivent être hautement disponibles. Elles peuvent être de différents types comme un système de fichiers, une adresse IP virtuelle, un ensemble de règles pour un pare-feu, un service réseau (Web, messagerie), un SGBD...

Sur chaque ressource, on peut appliquer les opérations *start*, *stop*, *status* et *monitor* (cette dernière opération n'existait pas avec *heartbeat-1* ; elle permet un diagnostic plus précis que *status* sur l'état d'une ressource ; c'est grâce à cette opération que *heartbeat-2* peut effectuer des surveillances applicatives en lieu et place de *mon* par exemple).

La gestion des *Resources* est dévolue aux *Resource Agents*. Un *Resource Agent* est normalement un script *shell*, qui a un type et des paramètres. Quand on donne des valeurs aux paramètres, on obtient une *Resource Instance*, c'est-à-dire une véritable ressource exploitable.

Les *Resource Agents* peuvent être de 3 types :

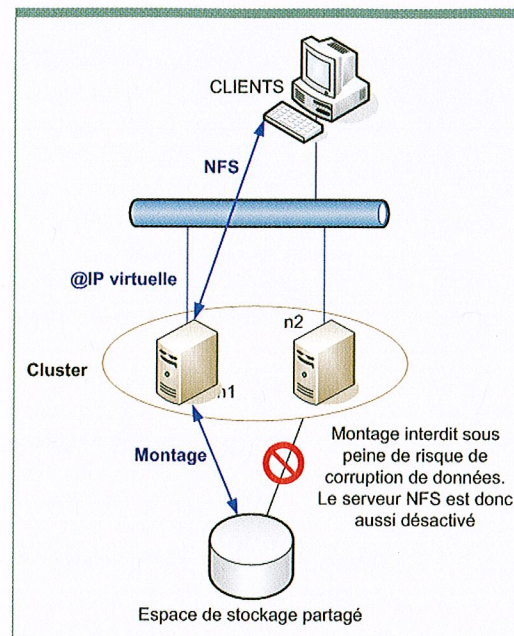
- le type *OCFResourceAgent* : indique des scripts qui se conforment à la norme OCF (*OpenCluster Framework* - voir [OCF]). *heartbeat-2* est livré avec un certain nombre de scripts. À l'installation, ils sont placés sous *\$OCF_ROOT/resource.d/heartbeat*.
- le type *LSBResourceAgent* : il s'agit des scripts classiques situés sous */etc/init.d*. Ces scripts doivent être conformes à la norme LSB (*Linux Standard Base*) (voir encart dans [LM97]).
- le type *HeartbeatResourceAgent* : ils ressemblent aux scripts précédents, mais diffèrent sur leur prise en compte de l'action *status* qui ne doit rien écrire si la ressource est arrêtée (voir <http://www.linux-ha.org/HeartbeatResourceAgent>). Ces scripts acceptent en outre des paramètres avant l'opération. Par exemple, dans le fichier *\$HA_DIR/ha.cf*, la ligne *IPaddr::192.168.1.155* provoquera l'appel suivant du script *IPaddr* ainsi : *IPaddr 192.168.1.155 start*.

Sur chaque *Resource Instance*, *heartbeat* peut appliquer les opérations *start*, *stop* et *status*. Si la ressource est de type *OCFResourceAgent*, l'opération *metadata* est en outre possible. Elle retourne une auto-description XML de la ressource. Cependant, si l'option *crm yes* n'a pas été indiquée dans le fichier *ha.cf*, le type *OCFResourceAgent* n'est pas supporté.

6.3 Vite un exemple !

Je vous propose de réaliser un script de ressource simple au format OCF : nous allons créer une ressource qui active ou désactive des exportations NFS.

Le scénario d'utilisation est le suivant : tous les nœuds peuvent accéder à un espace de stockage partagé, accessible via iSCSI ou via un SAN. Mais le système de fichiers qui gère les données de cet espace n'est pas distribué et ne gère pas les accès concurrents. Il est de type *ext3* par exemple. Or, nous voulons rendre accessible ce système de fichiers à des clients NFS.



Il faut donc qu'un seul nœud accède à cet espace de stockage, puis l'exporte par NFS. Pour obtenir ce résultat, il faut tout d'abord que le nœud « monte » l'espace de stockage, puis, dans une 2ème étape, exporte le système de fichiers par NFS.

La ressource *Filesystem* gère la 1ère étape. Nous allons donc créer un script qui ne s'occupera que de l'exportation par NFS. Nous créons le fichier *NFSserver* sous *\$OCF_ROOT/resource.d/heartbeat*.

Étape 1 : choix des paramètres du script

Nous définissons les paramètres suivants :

Nom du paramètre	Obligatoire	Description
export	oui	Nom du répertoire à exporter
client	oui	Spécification des clients autorisés à accéder à l'exportation, par exemple : <i>192.168.1.0/24</i>
options	non	Options de montage, au format du fichier <i>/etc/exports</i> . Par exemple, <i>rw, sync</i> pour donner un accès en lecture/écriture pour notre LAN

La valeur de chaque paramètre est passée au script par *heartbeat* via des variables dont le nom sera *OCF_RESKEY_export*, *OCF_RESKEY_client* et *OCF_RESKEY_options*. Ce tableau nous permet de coder la description des métadonnées de la ressource. Le début du script pourrait être :

```
#!/bin/sh
#
# Liste des paramètres:
#   OCF_RESKEY_export
#   OCF_RESKEY_client
#   OCF_RESKEY_options
#
```

```
# Initialization:
. ${OCF_ROOT}/resource.d/heartbeat/.ocf-shellfuncs

usage() {
    # Afficher les informations sur la syntaxe d'appel...
    echo "Usage..."
}

# Description des paramètres. Ils seront disponibles à travers
# l'interface graphique.
NFSservermeta_data() {
    cat <<END
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="NFSserver">
<version>1.0</version>

<longdesc lang="en">
Resource script for NFSserver. It manages a NFS export as an HA resource.
</longdesc>
<shortdesc lang="en">NFSserver resource agent</shortdesc>

<parameters>
<parameter name="export" unique="1" required="1">
<longdesc lang="en">Directory to export</longdesc>
<shortdesc lang="en">Directory to export</shortdesc>
<content type="string" default="/" />
</parameter>

<parameter name="client" unique="1" required="1">
<longdesc lang="en">Client specification</longdesc>
<shortdesc lang="en">Client specification</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="options" unique="1" required="0">
<longdesc lang="en">Mount options as specified in /etc/exports</longdesc>
<shortdesc lang="en">Mount options</shortdesc>
<content type="string" default="rw,async" />
</parameter>
</parameters>

# Nous anticipons un peu et définissons dès maintenant les actions
# qu'on peut effectuer sur cette ressource :
<actions>
<action name="start" timeout="30" />
<action name="stop" timeout="30" />
<action name="reload" timeout="30" />
<action name="status" timeout="30" />
<action name="monitor" depth="0" timeout="30" interval="10" start-delay="10" />
<action name="meta-data" timeout="5" />
<action name="validate-all" timeout="5" />
</actions>
</resource-agent>
END
}
```

Étape 2 : écriture des actions

Chaque action est implémentée par une fonction. Le script doit retourner un code retour standardisé, défini dans le fichier *\$OCF_ROOT/resource.d/heartbeat/.ocf-returncodes*. Les fonctions (simplifiées) pourraient ressembler à :

```
# Fonction interne - test de bon fonctionnement simple
NFSserver_status() {
    showmount -e localhost | grep "$OCF_RESKEY_export" >/dev/null 2>&1
    return $?
}
```

```
# Appelée avec le paramètre "status"
NFSServer_report_status() {
    if NFSServer_status; then
        # ocf_log est situé dans $OCF_ROOT/resource.d/heartbeat/ocf-shellfuncs
        ocf_log info "NFS Export $OCF_RESKEY_export is exported"
    else
        ocf_log info "NFS Export $OCF_RESKEY_export is not exported"
        return $OCF_NOT_RUNNING
    fi
    return $OCF_SUCCESS
}

# Plus le test est précis, mieux fonctionnera le Cluster:
NFSServer_monitor() {
    if NFSServer_status; then
        : OK
    else
        ocf_log info "NFS Export $OCF_RESKEY_export is not exported"
        return $OCF_NOT_RUNNING
    fi

    # Ici, on peut mettre des test applicatifs plus poussés
    return $OCF_SUCCESS
}

NFSServer_start() {
    ocf_log info "Activating NFS Export $OCF_RESKEY_export"

    exportfs -i -o $OCF_RESKEY_options $OCF_RESKEY_client:$OCF_RESKEY_export >/dev/null 2>&1

    if [ $? -eq 0 ]; then
        : OK Export $OCF_RESKEY_export activated just fine!
        return $OCF_SUCCESS
    else
        ocf_log err "NFS Export: $OCF_RESKEY_export did not activate correctly"
        return $OCF_ERR_GENERIC
    fi
}

NFSServer_stop() {
    ocf_log info "Deactivating NFS Export $OCF_RESKEY_export"
    exportfs -u $OCF_RESKEY_client:$OCF_RESKEY_export >/dev/null 2>&1
    if [ $? -ne 0 ]; then
        ocf_log err "NFS Export: $OCF_RESKEY_export did not stop correctly"
        return $OCF_ERR_GENERIC
    fi
    return $OCF_SUCCESS
}

# Vérifie la validité des paramètres OCF_RESKEY_*
NFSServer_validate_all() {
    if [ "$OCF_RESKEY_export" = "" ]; then
        ocf_log err "Directory to export not specified !"
        exit $OCF_ERR_GENERIC
    fi

    if [ ! -d "$OCF_RESKEY_export" ]; then
        ocf_log err "$OCF_RESKEY_export is not a valid Directory !"
        exit $OCF_ERR_GENERIC
    fi
    return $OCF_SUCCESS
}
```

Étape 3 : écriture de la partie principale

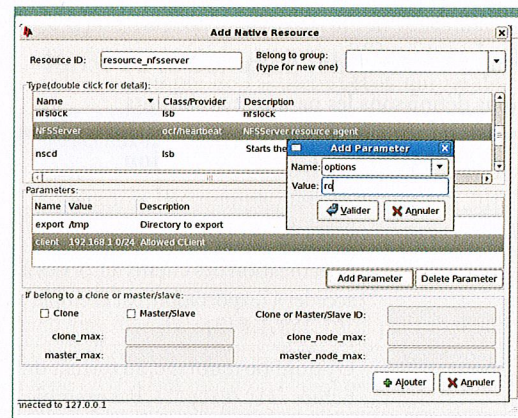
Blindons le script à minima... Puis, en fonction de la valeur de l'action, il faut appeler la fonction précédente adéquate.

```
if [ $# -ne 1 ]; then
    usage
    ocf_log err "Action Parameter missing"
    exit $OCF_ERR_ARGS
fi

# What kind of method was invoked?
case "$1" in
    meta-data) NFSServer_meta_data
                exit $OCF_SUCCESS;;
    start)      NFSServer_start
                exit $?;;
    stop)       NFSServer_stop
                exit $?;;
    reload)     NFSServer_stop
                NFSServer_start
                exit $?;;
    status)     NFSServer_report_status
                exit $?;;
    monitor)    NFSServer_monitor
                exit $?;;
    validate-all) NFSServer_validate_all
                ;;
    *)          usage
                exit $OCF_ERR_ARGS;;
esac
```

Étape 4 : tests du script

Avec l'interface graphique **hb_gui**, on crée une nouvelle ressource : le type **NFSServer** apparaît et les paramètres obligatoires **export** et **client** sont d'emblée proposés. Dans l'exemple ci-dessous, nous décidons d'exporter le répertoire **/tmp** en lecture-seule pour le réseau local :



On peut aussi utiliser le script **ocf-tester** fourni avec **heartbeat-2**. La syntaxe d'invocation serait ici :

```
# ocf-tester -n resource_nfsserver -o export=/tmp \
-o client=192.168.1.0/24 \
-o options=ro \
/usr/lib/ocf/resource.d/heartbeat/NFSServer
```

Ce script simule un appel du script **NFSServer** par **heartbeat** en créant les bonnes variables d'environnement, puis on l'invoque en lui passant les paramètres indiqués par les options **-o**. Il effectue des appels successifs avec les actions **meta-data**, **validate-all**, puis enchaîne une suite de **start**, **stop** et **monitor**.

Si aucune erreur n'apparaît, n'oubliez pas de recopier le script sur tous les nœuds du cluster !

6.4 Intégration dans un cas concret

Imaginons maintenant la réalité : vous voulez toujours rendre accessible par NFS un espace partagé à accès non concurrent. Quelles sont les ressources mises en œuvre ? Dans quel ordre faut-il les enchaîner ?

Tout d'abord, l'utilisation de **IPaddr2** permet de gérer l'adresse IP virtuelle référencée par les clients NFS. Puis, si le système de fichiers est géré par le LVM, il faut le démarrer grâce au script **LVM**. Ensuite, il faut monter le volume logique sous un répertoire de travail, la ressource **Filesystem** sait faire cela. Enfin, le script **NFSServer** peut exporter ce répertoire.

Cela nous fait, dans cet exemple, 4 ressources à décrire puis à enchaîner dans le bon ordre. Si vous avez suivi les étapes précédentes, cela devrait être un jeu d'enfant, en utilisant **hb_gui** !

7. Gestion des ressources

Si l'interface graphique **hb_gui** gère le fichier des ressources **\$HA_VARLIB/crm/cib.xml** et dialogue avec le DC, il peut s'avérer utile de réaliser ces opérations en mode ligne de commande.

Par exemple, les capacités de remontées d'erreur sont nulles avec l'interface graphique, et, si un problème survient, vous serez obligé d'utiliser les outils suivants et d'explorer vous-même les logs. Ce qu'on appelle déboguer quoi !

7.1 Manipuler le fichier de configuration

La commande **crm_sh** permet d'extraire certaines informations du fichier de ressources. Sans argument, elle fonctionne à la manière d'un shell :

```
n1 # crm_sh
crm # help
Usage: crm (nodes|config|resources)
crm # nodes
crm nodes # help
Usage: nodes (status|list)
crm nodes # list
<node id="f3643212-9c27-4f11-88b9-83f5eb" uname="n2.zereso.org" type="normal"/>
<node id="a328e624-2655-47d5-8c38-5c90f0" uname="n1.zereso.org" type="normal"/>
crm nodes # ^D
```

avec des arguments, elle affiche le résultat de la commande :

```
n1 # crm_sh resources status
ipaddr_test (heartbeat::ocf:IPaddr2)
n1 #
```

La commande **cibadmin** permet aussi de manipuler le fichier XML **cib.xml** pour en extraire ou y intégrer de nouvelles ressources. Cette commande interroge le DC (*Designated Coordinator*) pour requérir le fichier de ressources. Elle peut donc être jouée sur n'importe quel nœud :

```
# cibadmin -Q
```

Affiche le fichier de ressources. Tandis que la commande suivante

```
# cibadmin -Q -m
CIB on localhost_is_ the master instance
```

indique si la machine locale est ou n'est pas le DC. Il est possible de modifier, détruire ou ajouter des nœuds au fichier de ressources. Par exemple, pour ajouter une nouvelle ressource, il suffit de créer un fichier, par exemple **ressource1.xml** contenant le fragment XML suivant :

```
<primitive id="identifiant_unique">
  <instance_attributes id="RA_identifiant_unique">
    <attributes>
      <nvpair id="stop_identifiant_unique" name="target_role" value="Stopped"/>
    </attributes>
  </instance_attributes>
</primitive>
```

Puis, la commande suivante ajoute le contenu du fichier à la base de ressources. La modification est propagée sur tous les nœuds par le DC :

```
# cibadmin --obj_type resources -U -x ressource1.xml
```

La commande suivante permet de supprimer la ressource nouvellement créée :

```
# cibadmin -D -X '<primitive id="identifiant_unique" />'
```

Si vous faites des modifications manuellement dans le fichier **cib.xml** – mais nous ne le recommandons pas –, utilisez la commande **ciblint** pour vérifier la validité du nouveau fichier. Sur un fichier erroné, voici un exemple des erreurs qui peuvent être remontées :

```
# ciblint -f cib.xml
ERROR: <nvpair> id '930da5e5-26ed-46f2-8014-ba034d50e850' not unique among all <nvpair> tags
ERROR: name 'nic' multiply defined in <nvpair> list - values are [eth0] and [eth0]
ERROR: <nvpair> id '930da5e5-26ed-46f2-8014-ba034d50e850' multiply defined in <nvpair> list
WARNING: STONITH disabled <nvpair name="stonith-enabled" value="false">. STONITH is STRONGLY recommended.
WARNING: No STONITH resources configured. STONITH is not available.
```

Ici, nous avons un nœud **<nvpair>** définissant deux fois l'interface **eth0** et la fonctionnalité de **STONITH** est désactivée.

7.2 Gestion du cluster

7.2.1 Gestion dynamique des ressources

La commande **crm_resource**, évoquée brièvement au §5.4, permet d'interagir avec le gestionnaire du cluster. Elle permet :

► d'obtenir la liste des ressources :

```
# crm_resource -L
Resource Group: group_ipaddr_smb
resource_ipaddr2 (heartbeat::ocf:IPaddr2)
resource_smb (heartbeat::ocf:Filesystem)
resource_nfsserver (heartbeat::ocf:NFSServer)
```

► de connaître sur quel nœud est localisé une ressource :

```
# crm_resource -W -r resource_smb
resource resource_smb is running on: n1.zereso.org
```

► de migrer une ressource d'un nœud vers un autre (voir §5.4) :

```
# crm_resource -M -r resource_smb -H n2.zereso.org
```

► de détruire une ressource :

```
# crm_resource -D -r resource_smb -t primitive
```

► d'obtenir et modifier les propriétés d'une ressource :

```
# crm_resource -Q -r resource_smb
```

7.2.2 Mise en stand-by des nœuds

La commande `crm_standby` interagit sur les nœuds eux-mêmes pour les mettre en mode *stand-by* ou bien les réactiver. Cela permet d'effectuer des opérations de maintenance sur un nœud, en toute sérénité. La mise en stand-by peut être définitive (*forever*), valable jusqu'au prochain redémarrage (*reboot*) ou bien levée quand on veut :

► par exemple, pour mettre n1 en stand-by :

```
# crm_standby -v true -U n1.zereso.org
# crm_mon -l
...
Node: n1.zereso.org (a328e624-2655-47d5-8c38-5f7ccb2c90f0): standby
```

► pour obtenir le statut de n1 :

```
# crm_standby -G -U n1.zereso.org
scope=nodes value=true
```

► pour réactiver n1 :

```
# crm_standby -G -U n1.zereso.org
```

7.2.3 Ajout et suppression de nœuds

Nous avons indiqué en préambule que `heartbeat-2` permettait la gestion de clusters composés de plus de 2 nœuds.

Comment ajouter un nœud supplémentaire ?

L'interface graphique ne permet pas (encore) de la faire. Où déclare-t-on les nœuds ? Dans le fichier `$HA_DIR/ha.cf`. On va donc procéder comme suit :

- On détermine qui est le DC, à l'aide de la commande `crm_mon -i 1`.
- On ajoute le nouveau nœud dans le fichier `$HA_DIR/ha.cf` du DC.
- Il faut recopier le fichier modifié sur tous les nœuds du cluster. Comme c'est pénible, on peut utiliser le script `ha_propagate` qui boucle sur la liste des nœuds et y transfère les fichiers `ha.cf` et `authkeys`. Attention, sur tous les nœuds, `heartbeat` doit avoir été installé à l'identique et un serveur `ssh` doit être activé.
- Sur le nouveau nœud, assurez-vous que `heartbeat` a été démarré...

Comment supprimer un nœud ?

Une fois en activité, le nœud est référencé non seulement dans le fichier `$HA_DIR/ha.cf`, mais aussi dans la CIB ! Il faut donc utiliser la commande `cibadmin` (vue au §7.1) pour supprimer sa définition dans le fichier `cib.xml` :

```
# cibadmin -D -o nodes -X '<node id="....." ..... />'
```

Puis, on supprimera sa référence dans le fichier `$HA_DIR/ha.cf` du DC et on propagera la modification comme précédemment, à l'aide de la commande `ha_propagate`.

8. Pour aller plus loin

Cette approche mi-graphique, mi-ligne de commande de `heartbeat-2` est forcément incomplète. De nombreux détails ont été passés sous silence comme certaines techniques communes à toutes les technologies de *clustering* :

- le fencing – mise à l'écart des nœuds invalides – ;
- la résolution du problème du *split-brain*, qui surgit typiquement dans un cluster à 2 nœuds où chacun croit qu'il doit acquérir la ressource et que l'autre est indisponible ;
- la gestion du *quorum*, pour les clusters à plus de 2 nœuds : il s'agit d'un système de votes qui permet de décider si un nœud est invalide ou non, puis qui doit prendre le rôle du DC.

`heartbeat-2` ajoute aussi son propre lot de fonctionnalités comme :

- la possibilité de définir des ressources **clonables** ; ainsi, une même ressource peut être démarrée sur « n » nœuds simultanément, utilisables pour les systèmes de fichiers « clusterisés », les clusters avec équilibre de charge, toute architecture où les ressources sont déployées de manière multiple et simultanée ;
- la possibilité de définir des ressources possédant un **sous-état maître ou esclave**, pour des ressources comme les bases de données répliquées, DRBD...

Par sa conception et sa gestion des ressources, `heartbeat-2` ressemble fort au *RedHat Cluster* dans sa version 4 (pour ceux qui connaissent). Cependant, même s'il est livré avec de nombreux scripts de gestion de ressources, `heartbeat-2` n'intègre pas de systèmes de fichiers distribués « à la GFS ». De plus, son interface graphique est certes utile pour créer une configuration initiale, mais reste limitée : pas de glisser/déposer pour migrer les ressources, pas de possibilité d'ajouter ou supprimer des nœuds, pas de remontée efficace des messages d'erreur... Le projet évolue toutefois en permanence et l'accent n'est pas forcément mis sur l'interface graphique. À suivre donc...

Outre les documents disponibles sur le site [HRBT], notons que Novell, éditeur de la distribution SuSE a fait un bon travail de documentation sur son site [SLES].

Jérôme Delamarche,

jdelamarche@maje.biz



RÉFÉRENCES

- [NTP] <http://tldp.org/HOWTO/TimePrecision-HOWTO/ntp.html>
- [HRBT] <http://www.linux-ha.org>
- [OCF] <http://opencf.org>
- [LM97] *GNU/Linux Magazine* n°97
- [TUT0] <http://www.linux-ha.org/HeartbeatTutorials>
- [SLES] <http://www.novell.com/documentation/sles10/heartbeat/>