Haute disponibilité d'un service Web dynamique

Activité 1 - Installation et première configuration des serveurs primaire (serveur maître) et secondaire (serveur esclave)

L'objectif opérationnel est ici de préparer les deux serveurs à l'identique en installant et en procédant à une première configuration des outils de haute disponibilité.

Vous disposez, sous VirtualBox, d'une machine virtuelle (et vous êtes en mesure de la cloner) sur laquelle se trouve l'application Web opérationnelle en HTTP ou HTTPS.

Vous disposez de la documentation suivante :

- **Document 1 :** fonctionnement et installation de Corosync et Pacemaker
- **Document 2** : configuration de Corosync
- **Document 3** : commandes utiles et configuration du cluster

Travail à faire

Le travail sera dans un premier temps réalisé sur le serveur qui tiendra lieu de serveur maître. C'est ce serveur qui sera cloné par la suite.
Le serveur maître aura pour nom d'hôte « serv1 » et le serveur secondaire « serv2 »).

Q1. Vérifiez que l'application *Gestion de frais* est bien accessible sur le serveur maître via son adresse IP et son nom d'hôte.

Q2. A l'aide du document 1, installez *Corosync* et *Pacemaker* et procédez aux configurations nécessaires.

Q3. Démarrez le service (éventuellement après avoir relancer la machine) et vérifiez que *Corosync* est opérationnel.

Q4. Clonez le serveur primaire et procédez à la configuration IP du serveur esclave.

Q5. Démarrez les nœuds et vérifiez que *Corosync* est opérationnel sur les deux nœuds.

Q6. Vérifiez la configuration de *Corosync* et désactivez STONITH et le quorum.

Q7. Vérifiez de nouveau la configuration ainsi que le fichier XML généré.

Documentation

Document 1 : fonctionnement et installation de Corosync et Pacemaker

Un **cluster** est un groupe d'ordinateurs reliés qui travaillent en étroite collaboration. Un cluster est donc constitué d'au moins 2 machines (physiques ou virtuelles).
Les machines d'un cluster sont appelées **« nœuds »** (*nodes* en anglais). Un nœud est donc une machine qui participe à un cluster (qui est membre d'un cluster).

Les clusters sont basés sur le principe des battements de cœur. Un « heartbeat » (battement de cœur, en anglais), est le fait qu'un nœud puisse entendre le « cœur » d'un autre nœud battre. Ainsi, un nœud peut savoir si les autres nœuds sont encore vivant ou non. Lorsque le « cœur » d'un autre nœud ne bat plus, on considère qu'il est mort et des actions doivent être entreprises. Nous installerons :

- ***Corosync*** qui jouera ce rôle en gérant, via des échanges d'informations entre les nœuds, l'infrastructure de cluster, c'est à dire l'état des nœuds et leur fonctionnement en groupe ;
- ***Pacemaker*** (stimulateur cardiaque) qui est un gestionnaire de cluster haute disponibilité. Il est chargé de créer, démarrer, arrêter et superviser les ressources du cluster c'est à dire les services gérés en cluster et donc inclus dans la continuité de services.

Dans son mode le plus simple, cette solution de clustering est basée sur le modèle actif-passif : lorsqu'un serveur considéré comme « primaire » n'est plus considéré comme accessible par *Corosync*, *Pacemaker* démarre sur un serveur secondaire les services qu'on lui indique dans la configuration.

Gérer un service en cluster (un service étant une ressource au sens de Pacemaker) consiste donc, dans sa forme la plus simple (mode actif/passif), à installer le service sur tous les nœuds participants au cluster et à n'activer ce service que sur le nœud dit maître (nœud actif). Lorsque ce nœud est défaillant, le service est activé automatiquement sur le nœud dit esclave (nœud passif qui devient actif). Le retour automatique du service (auto failback) sur le nœud maître (lorsque celui-ci est de nouveau en état) n'est pas systématique et doit être géré au niveau de la configuration de chaque ressource.

Mette à jour la liste des paquet : **apt update**

Et installer les outils nécessaires : **apt install corosync pacemaker crmsh**

Le service corosync est lancé : **root@serv1:~# service corosync status**

```
● corosync.service - Corosync Cluster Engine
   Loaded: loaded (/lib/systemd/system/corosync.service; enabled)
   Active: active (running) since mer. 2016-07-20 20:43:06 CEST; 22s ago
 Main PID: 4792 (corosync)
   CGroup: /system.slice/corosync.service
           └─4792 /usr/sbin/corosync -f

juil. 20 20:43:06 serv1 corosync[4792]: [WD    ] no resources configured.
...
juil. 20 20:43:06 serv1 corosync[4792]: [QUORUM] Members[1]: 2130706433
```

Mais il est nécessaire de le configurer en ajoutant notamment les nœuds du cluster.

Document 2 : configuration de Corosync

L'exemple de configuration qui suit est basé sur la configuration IP suivante :

serv1: serveur maître
- IP réelle : 172.16.0.10 --> c'est celle qui est dans /etc/network/interfaces
- IP virtuelle : 172.16.0.12

serv2 : serveur esclave
- IP réelle : 172.16.0.11 --> c'est celle qui est dans /etc/network/interfaces
- IP virtuelle : 172.16.0.12

En production, il est fortement conseillé d'utiliser 2 interfaces réseau, une dédiée à l'échange d'information sur le cluster entre les nœuds. Par mesure de simplification, nous n'utiliserons dans notre labo de test qu'une seule interface.

Étape 1

Pour gérer l'infrastructure de cluster (état des nœuds et leur fonctionnement en groupe), il est nécessaire de générer une clef d'authentification qui sera partagée par tous les nœuds du cluster. Cette clé servira aussi à chiffrer les échanges entre les nœuds. L'utilitaire *corosync-keygen* permet de générer cette clef à partir d'entrées clavier pseudo-aléatoires.

root@serv1:~# corosync-keygen

```
Corosync Cluster Engine Authentication key generator.
Gathering 1024 bits for key from /dev/random.
Press keys on your keyboard to generate entropy.
Press keys on your keyboard to generate entropy (bits = 152).
Press keys on your keyboard to generate entropy (bits = 200).
…
...
Press keys on your keyboard to generate entropy (bits = 968).
Press keys on your keyboard to generate entropy (bits = 1016).
Writing corosync key to /etc/corosync/authkey.
```

Lorsque l'on est sur une console à distance et de manière à générer le maximum de caractères, l'astuce consiste à ouvrir une autre console et à mettre à jour la liste des paquets puis à installer des paquets (par exemple nmap, tcpdump et lynx qui pourront nous être utiles).

Le fichier authkey a bien été créé dans /etc/corosync avec des droits restreints : **ls -l /etc/corosync/**

```
-r-------- 1 root root  128 juil. 20 22:22 authkey
-rw-r--r-- 1 root root 3929 févr. 18 16:18 corosync.conf
```

Ce fichier doit être copié sur tous les autres nœuds du cluster dans chaque dossier /etc/corosync. Dans notre cas, l'autre nœud sera créé par clonage, il contiendra donc le fichier.

Étape 2

Le fichier de configuration est **/etc/corosync/corosync.conf**. Les valeurs par défaut de nombreuses directives conviennent, nous ne les modifierons pas mais comme le fichier d'origine comporte de nombreux commentaires, nous créerons notre propre fichier (après avoir sauvegardé le fichier d'origine) avec les directives principales.

Sauvegarde du fichier d'origine : mv /etc/corosync/corosync.conf /etc/corosync/corosync.conf.ori

Création d'un nouveau fichier : vi /etc/corosync/corosync.conf

Directives et valeurs	Explications

`totem {`
`  version: 2`
`  cluster_name: cluster_web`
`  crypto_cipher: aes256`
`  crypto_hash: sha1`
`  clear_node_high_bit:yes`
`}`
	De manière à ce que les échanges entre les nœuds soient chiffrés par la clé préalablement générée, on définit les algorithmes de chiffrement et de hachage utilisés.

`logging {`
`  fileline: off`
`  to_logfile: yes`
`  logfile: /var/log/corosync/corosync.log`
`  to_syslog: no`
`  debug: off`
`  timestamp: on`
`  logger_subsys {`
`  subsys: QUORUM`
`  debug: off`
`  }`
`}`
	Gestion des logs : il serait plus judicieux d'utiliser un système de gestion de log avec démon mais par mesure de simplification, nous utiliserons la méthode classique avec un fichier de log local.

`quorum {`
`  provider: corosync_votequorum`
`  expected_votes: 2`
`  two_nodes: 1`
`}`
	Gestion du quorum (nombre minimum de nœuds en ligne pour être capable de valider une décision) : dans le cas d'un cluster avec Pacemaker, il faut que plus de la moitié des nœuds soit en ligne. Avec un cluster à deux nœuds, il n'y a plus de quorum dès qu'un nœud est perdu. Il faudra donc demander à Pacemaker d'ignorer le quorum dans cette situation car le fonctionnement par défaut, quand le quorum n'est pas atteint, est de couper toutes les ressources (voir par la suite).

`nodelist {`
`  node {`
`  name: serv1`
`  nodeid: 1`
`  ring0_addr: 172.16.0.10`
`  }`
`  node {`
`  name: serv2`
`  nodeid: 2`
`  ring0_addr: 172.16.0.11`
`  }`
`}`
	Déclaration des nœuds du cluster

`service {`
`  ver: 0`
`  name: pacemaker`
`}`
	Ajoute le service Pacemaker dans le cluster Corosync. La ligne "ver" définit le mode de démarrage de Pacemaker. Placé sur 1, Pacemaker devra être démarré manuellement ou en tant que service, après le démarrage de Corosync. Si la valeur est à 0 (valeur par défaut), Pacemaker est démarré (ou arrêté) par Corosync.

*D'autres interfaces peuvent être définies. Il suffit de définir un deuxième bloc interface {} Dans ce cas, *ringnumber* doit être incrémenté et l'adresse et le port multicast doivent être différents.

** Corosync propose le concept d'anneaux de connexion pour assurer la communication entre nœuds et permet de définir les anneaux en terme de réseau IP plutôt que de définir les adresses IP. C'est appréciable car le même fichier de configuration peut être déployé sur tous les nœuds sans rien changer ==> c'est ce que nous allons faire via le clonage de la VM.

À noter que d'autres configurations sont possibles, notamment en utilisant l'unicast plutôt que le multicast auquel cas il faut ajouter la directive transport: udpu et spécifier un nodelist avec les adresses IP statiques des nœuds.

La configuration peut être vérifiée avec la commande suivante :
root@serv1:~# corosync-cfgtool -s

```
Printing ring status.
Local node ID 1084751984
RING ID 0
        id      = 127.0.0.1
        status  = ring 0 active with no faults
```

La configuration du cluster est contenue dans un cluster Information Base (CIB) qui est un fichier XML. **/var/lib/pacemaker/cib/cib.xml :** ce fichier (voir le contenu dans le document 3) est généré à partir du fichier de configuration et est complété au fur et à mesure des ajouts de ressources.

Ce dernier fichier ne doit JAMAIS être modifié directement mais via les commandes vues ci-après.

En cas de problème, vous devez lire attentivement les messages d'erreurs et vous ne devez pas hésiter à consulter les logs !

Document 3 : commandes utiles et configuration du cluster

Pour vérifier que *corosync* est lancé : */etc/init.d/corosync status* (ou ***service corosync status)***

La commande crm implémentée par pacemaker (qui peut être avantageusement utilisée aussi en interactif) permet de gérer la configuration et les différentes ressources. Elle **se propage sur chaque nœud automatiquement** : **la commande s'effectue donc sur n'importe quel nœud.**

Avant toute chose, un petit aperçu des commandes importantes pour vérifier l'état du cluster :

La commande suivante **crm_mon** permet de suivre l'**évolution de la configuration en direct :**

```
Last updated: Thu Jul 21 10:07:43 2016          Last change: Thu Jul 21 02:20:18 2016 by hacluster via
crmd on serv2
Stack: corosync
Current DC: serv1 (version 1.1.14-70404b0) - partition with quorum
2 nodes and 0 resources configured

Online: [ serv2 serv1 ]
```

On sort de cette commande avec [CTRL + C]. **Le paramètre -1 permet d'obtenir une seule sortie de même que la commande crm status.**
Cette commande remonte un certain nombre d'informations sur l'état général du cluster (nœuds ok, hs, offline ou en standby) et sur l'état des ressources (quel nœud gère quelle ressource, les éventuels problèmes rencontrés lors du lancement ou du monitoring des ressources…).

Last updated et stack : la dernière mise à jour des informations du moniteur et le fournisseur (provider) utilisé : ici, corosync
Current DC : c'est le nœud qui va coordonner les actions sur le cluster. Les autres nœuds remonteront leurs informations au nœud DC. Ce nœud n'est pas nécessairement le nœud « maître » du cluster.
Nodes : le nombre de nœuds participant au cluster.
Resources configured : le nombre de ressources configurées dans le cluster.
Online : la liste des nœuds **online**. On pourra aussi trouver ici, la liste des nœuds **offline** ou en **stand-by.**

Le contrôle d'un service actif (ou pas) sur un nœud peut aussi s'effectuer via les commandes usuelles ***ps***, ***netstat*** et ***nmap.***

Pour afficher la configuration (beaucoup plus lisible que le fichier xml) :

```
root@serv1:~# crm configure show
node 1084751984: serv1
node 1084751987: serv2
property cib-bootstrap-options: \
        have-watchdog=false \
        dc-version=1.1.14-70404b0 \
        cluster-infrastructure=corosync \
        cluster-name=debian
```

Le fichier XML correspondant est le suivant :

```
<cib crm_feature_set="3.0.10" validate-with="pacemaker-2.4" epoch="5" num_updates="0" admin_epoch="0" cib-last-written="Thu Jul 21 02:26:58 2016" update-origin="serv2" update-client="crmd" update-user="hacluster" have-quorum="1" dc-uuid="1084751984">
  <configuration>
    <crm_config>
      <cluster_property_set id="cib-bootstrap-options">
        <nvpair id="cib-bootstrap-options-have-watchdog" name="have-watchdog" value="false"/>
        <nvpair id="cib-bootstrap-options-dc-version" name="dc-version" value="1.1.14-70404b0"/>
        <nvpair id="cib-bootstrap-options-cluster-infrastructure" name="cluster-infrastructure" value="corosync"/>
        <nvpair id="cib-bootstrap-options-cluster-name" name="cluster-name" value="debian"/>
      </cluster_property_set>
    </crm_config>
    <nodes>
      <node id="1084751987" uname="serv2"/>
      <node id="1084751984" uname="serv1"/>
    </nodes>
    <resources/>
    <constraints/>
  </configuration>
</cib>
```

Désactivation de stonith

Il existe une commande qui permet de vérifier la validité du fichier de configuration XML alors généré :***crm_verify -L -V .***

À la première exécution de cette commande (après le démarrage correct de Corosync), des erreurs sont remontées :

root@serv1:~# crm_verify -L -V

```
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
```

« Stonith » (shot the other node in the head) permet de tuer proprement les nœuds qui sont morts.
C'est en fait un mécanisme pour éteindre complètement le serveur qui vient de flancher en éteignant son onduleur. C'est surtout utilisé avec des disques partagés car il serait dangereux que l'ordinateur qui est supposé être hors d'état vienne écrire sur le disque partagé et corrompre/altérer les données.

Par mesure de simplification, pour cette séance, nous allons désactiver STONITH par la commande : **crm configure property stonith-enabled=false**

root@serv1:~# crm configure property stonith-enabled=false

INFO: building help index

La vérification ne renvoie plus d'erreur. Vous pouvez consulter le fichier XML généré, ***cat /var/lib/pacemaker/cib/cib.xml*** :

```
<cib crm_feature_set="3.0.10" validate-with="pacemaker-2.4" epoch="6" num_updates="0" admin_epoch="0" cib-last-written="Thu Jul 21 10:44:58 2016" update-origin="serv1" update-client="cibadmin" update-user="root" have-quorum="1" dc-uuid="1084751984">
 <configuration>
  <crm_config>
   <cluster_property_set id="cib-bootstrap-options">
    ...
   <nvpair name="stonith-enabled" value="false" id="cib-bootstrap-options-stonith-enabled"/>
   </cluster_property_set>
  </crm_config>
 ...
 </configuration>
</cib>
```

Désactivation du Quorum

Il persiste encore un petit problème concernant le **quorum** qui est le nombre minimum de nœuds en ligne pour être capable de valider une décision. Dans le cas d'un cluster avec Pacemaker, il faut que plus de la moitié des nœuds soit en ligne. Avec un cluster à deux nœuds, il n'y a plus de quorum dès qu'un nœud est perdu. Il faut donc demander à Pacemaker d'ignorer le quorum dans cette situation car le fonctionnement par défaut, quand le quorum n'est pas atteint, est de couper toutes les ressources.

Pour désactiver le quorum, saisir la commande suivante :

crm configure property no-quorum-policy="ignore".