# Mise en cluster de l'applicatif

## Objectif

Le but de cette documentation est de présenter la mise en cluster d'un serveur Web avec la surveillance de la connectivité de la machine et du service nginx grace au service Corosync et Pacemaker.

## Pré-requis

- Un serveur Debian 7 opérationnel
- Deux cartes réseau, une pour le LAN et une pour la ligne de vie

## Installation

Nous commençons par configurer les adresse de nos deux cartes réseaux. Nous plaçons eth0 sur le réseau DMZ d'adresse 172.29.0.0/24 et eth1 pour la ligne de vie 10.29.0.0/30. Nous avons choisi de positionner notre premier serveur, GSB-Web, aux adresses 172.29.0.10/10.29.0.1 et le second serveur, GSB-Web2 aux adresses 172.29.0.11/10.29.0.2.

Nous installons ensuite pacemaker, contenant corosync comme dépendance puis, temporairement, le paquet **haveged**. Il nous permettra de créer de l'entropie pour la clé corosync que nous générerons ensuite :

```
# apt-get install pacemaker haveged
# corosync-keygen
# apt-get purge haveged
```

## Configuration

Une fois notre clé générée, nous la copions vers le second serveur et la plaçons dans le bon dossier, **/etc/corosync** :

```
GSB-Web ~ # scp /etc/corosync/authkey user@172.29.0.11:/home/user
GSB-Web2 ~ # mv /home/user/authkey /etc/corosync/
```

Une fois fait, nous modifions le fichier **/etc/corosync/corosync.conf** et plus précisement la valeur **bindnetaddr** contenu dans le bloc interface lui même contenu dans le bloc totem :

```
interface {
    # The following values need to be set based on your environment
    ringnumber: 0
    bindnetaddr: 172.29.0.0
```

```
    mcastaddr: 226.94.1.1
    mcastport: 5405
}
```

On ajoute notre second serveur dans le fichier **/etc/hosts** qui ressemblera à ceci :

```
127.0.0.1       localhost
172.29.0.10     GSB-Web
172.29.0.11     GSB-Web2

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Enfin, changeons la valeur **START** à **yes** dans le fichier **/etc/default/corosync** puis on démarre le service et vérifie l'état de notre cluster :

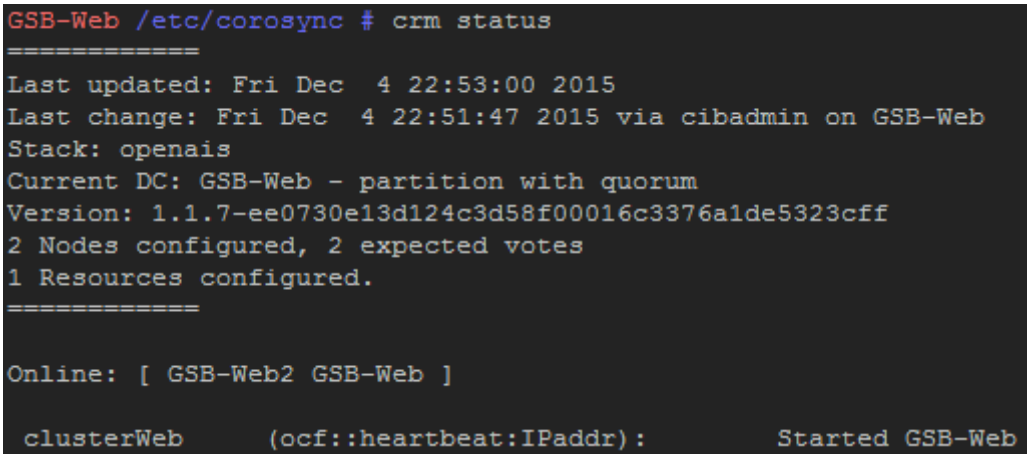
```
# service corosync start
# crm status
```

```
GSB-Web /etc/corosync # crm status
============
Last updated: Fri Dec  4 22:27:24 2015
Last change: Fri Dec  4 22:26:43 2015 via crmd on GSB-Web
Stack: openais
Current DC: GSB-Web2 - partition with quorum
Version: 1.1.7-ee0730e13d124c3d58f00016c3376a1de5323cff
2 Nodes configured, 2 expected votes
0 Resources configured.
============

Online: [ GSB-Web2 GSB-Web ]
```

A partir de maintenant, nous travaillons sur un seul noeud. On attribue une adresse IP virtuelle (172.29.0.100) à notre cluster en créant une primitive nommée clusterWeb. Nous finissons en vérifiant la configuration et en appliquant les changements si aucun problème n'est signalé :

```
crm
cib new config
configure
primitive clusterWeb ocf:heartbeat:IPaddr params ip="172.29.0.100"
cidr_netmask="24" nic="eth0" op monitor interval="5s"
property stonith-enabled=false
property no-quorum-policy=ignore
location preferred-GSB-Web clusterWeb 100: GSB-Web
verify
commit
end
```




Cette partie ne fonctionne que lorsque le serveur perd la connexion réseau. Nous allons maintenant configurer notre cluster pour gérer la panne du service web. Nous y ajouterons une colocation pour éviter la répartition des ressources entre les membres du cluster qui s'opére par défaut.

```
crm
configure
primitive serviceWeb ocf:heartbeat:nginx params
configfile="/etc/nginx/nginx.conf" meta migration-threshold="2" op monitor
interval="10s" op start timeout="30s" op stop timeout="30s"
colocation clusterWeb-serviceWeb inf: clusterWeb serviceWeb
commit
quit
```

La gestion du service web se faisant maintenant avec notre cluster, nous désactivons le démarrage automatique d'nginx

```
# update-rc.d -f nginx remove
```

# Mises à jour

## Pare-feu

On modifie la règle de redirection du pare-feu en modifiant l'IP de 172.29.0.10 à 172.29.0.100 :

```
# iptables -D PREROUTING -i eth2 -p tcp -m tcp --dport 80 -j DNAT --to-
destination 172.29.0.10:80
# iptables -A PREROUTING -i eth2 -p tcp -m tcp --dport 80 -j DNAT --to-
destination 172.29.0.100:80
```

## Nginx

Afin d'identifier facilement le serveur que notre cluster utilise, nous configurons un header qui sera renvoyé par nginx lors des requêtes des clients. Pour cela, nous modifions le fichier **/etc/nginx/sites-available/applis_gsb** et y ajoutons la ligne :

```
add_header "node" "WebX";
```

X correspondant au numèro de serveur.

# Jeu d'essai

| Situation | Opération(s) réalisée(s) | Résultat |
|---|---|---|
| Accès à l'applicatif (situation normale) | Rentrer l'IP du cluster dans un navigateur et on devrait obtenir la page de l'applicatif. | Depuis l'interface publique du routeur 172.16.2.2 et avec les bonnes redirections au niveau du pare-feu on obtiens bien la page de l'applicatif. |
| Arrêt du serveur maître | On coupe l'interface du serveur maître | On accède a l'application et lorsque l'on examine l'élément on voit bien que c'est le serveur 2 qui nous a répondu |
| Arrêt du serveur esclave | On coupe l'interface du serveur esclave | On accède a l'application et lorsque l'on examine l'élément on voit bien que c'est le serveur 1 qui nous a répondu |