Corosync/Pacemaker

Tamara Crétard - 20.11.2025

Sommaire:

1. Contexte

Dans le cadre du cours de BTS SIO SISR, l'objectif de ce projet est de comprendre et maîtriser la mise en place d'une infrastructure en haute disponibilité.

Pour cela, nous avons utilisé Corosync et Pacemaker, deux outils open-source de clustering qui permettent de rendre un service tolérant aux pannes matérielles ou logicielles. L'objectif final est de mettre en place un serveur web Apache hautement disponible, capable de basculer automatiquement sur un second nœud en cas de défaillance du premier.

2. Environnement de travail

Avant de commencer les différentes expérimentations avec Corosync et Pacemaker, il est essentiel d'analyser et de comprendre l'environnement de travail dans lequel le serveur et les machines clientes vont fonctionner.

2.1. Configuration utilisée

Tout d'abord, il est intéressant de connaître la configuration utilisée pour permettre l'installation des différentes technologies. Dans notre cas, la configuration utilisée est la suivante:

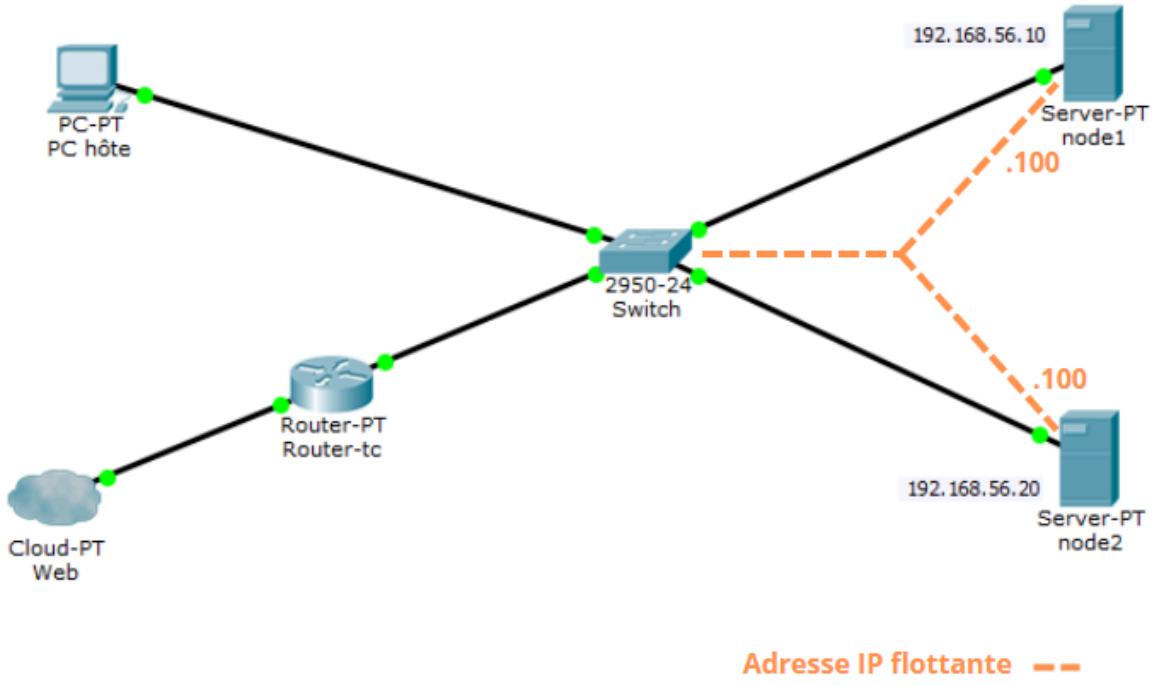
Processeur	11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz 3.60 GHz
RAM	32 Go
Carte graphique	NVIDIA GeForce RTX 3080
SSD	2 To
Système d'exploitation	Microsoft Windows 11 Famille
Architecture	x64

2.2. Schéma du réseau virtuel

Afin de mieux visualiser notre installation, un schéma du réseau virtuel est indispensable.

Dans notre infrastructure, nous avons besoin:

- d'ne machine hôte hébergeant les machines virtuelles
- de *node1* qui sera le premier nœud du cluster (192.168.56.10) en Réseau Privé Hôte
- de *node2* qui sera le second nœud du cluster (192.168.56.20) en Réseau Privé Hôte
- d'un routeur, avec une interface NAT et l'autre en Réseau Privé Hôte



3. Technologies et concepts utilisés

Différentes technologies et systèmes sont indispensables pour nos expérimentations. Il est donc nécessaire d'en comprendre le fonctionnement avant de les mettre en pratique.

3.1. Haute disponibilité

La Haute Disponibilité (High Availability en anglais) permet de garantir un accès continu aux services informatiques, 24 heures/24. Pour cela, il y a une redondance des équipements et des transferts automatiques sont mis en place en cas d'incident.

La Haute Disponibilité désigne le fait qu'une architecture ou un service a un taux de disponibilité convenable. C'est un enjeu important car une indisponibilité entraîne des coûts très élevés: des coûts car les clients n'ont pas pu faire leur commande et des coûts liés à l'image.

3.2. Failover

Le failover (basculement automatique en français) est le mécanisme qui permet à un cluster de continuer à fonctionner même quand un serveur tombe en panne. Le failover fonctionne avec une adresse IP virtuelle. Quand le serveur principal ne répond plus, le cluster détecte le problème en quelques secondes et bascule automatiquement tous les services vers un serveur de secours. L'adresse IP virtuelle se déplace aussi vers ce nouveau serveur, ce qui permet aux utilisateurs de continuer à accéder au service sans interruption. Ce processus est totalement automatique et transparent pour les utilisateurs qui ne remarquent souvent qu'un court ralentissement. Une fois le serveur en panne réparé, il peut reprendre sa place dans le cluster et redevenir actif. Le failover est donc essentiel pour garantir la haute disponibilité d'un service.

3.3. Apache

Apache est un serveur web open-source créé en 1995. C'est l'un des serveurs web les plus utilisés au monde. Apache permet de servir des pages web aux utilisateurs qui en font la demande via leur navigateur. Dans notre projet, Apache représente le service critique que nous souhaitons rendre hautement disponible.

3.4. Corosync

Corosync est un logiciel open-source créé en 2008 qui permet à plusieurs serveurs de communiquer entre eux dans un cluster. Un cluster est un groupe d'ordinateurs qui travaillent ensemble pour offrir un service sans interruption. Corosync s'occupe de faire communiquer les différents serveurs de manière fiable et rapide. Il vérifie aussi que suffisamment de serveurs fonctionnent correctement, ce qu'on appelle le quorum. Si un serveur tombe en panne, Corosync le détecte immédiatement et prévient les autres. Il utilise un protocole spécial appelé Totem qui garantit que tous les serveurs reçoivent les informations dans le même ordre, ce qui évite les problèmes de synchronisation.

3.5. Pacemaker

Pacemaker est un gestionnaire de cluster open-source créé en 2004. Il fonctionne avec Corosync et s'occupe de gérer les services du cluster, comme un site web ou une base de données. Quand un serveur tombe en panne, Pacemaker déplace automatiquement les services vers un autre serveur qui fonctionne bien. On peut aussi lui donner des règles pour décider où les services doivent tourner et dans quel ordre ils doivent démarrer. Pacemaker peut même déplacer des services d'un serveur à un autre sans que les utilisateurs s'en aperçoivent. Il prend ses décisions en fonction des informations que lui donne Corosync et des règles configurées par l'administrateur.

3.6. PCS (Pacemaker/Corosync Configuration System)

PCS est un outil en ligne de commande qui simplifie la configuration et la gestion des clusters Pacemaker et Corosync. Au lieu de modifier des fichiers de configuration complexes, PCS permet d'utiliser des commandes simples pour créer et gérer le cluster. On

peut facilement ajouter des serveurs au cluster, configurer les ressources, définir les règles de fonctionnement et surveiller l'état du cluster. C'est devenu l'outil standard pour administrer les clusters, notamment sur les systèmes Red Hat et CentOS. PCS rend la gestion des clusters beaucoup plus accessible aux administrateurs.

3.7. Concepts importants

Pour passer à la mise en œuvre de notre travail, il est aussi important de connaître certains concepts importants.

Le *quorum* est le nombre minimum de serveurs qui doivent fonctionner pour que le cluster reste actif. Dans un cluster à deux serveurs, on le fixe généralement à 1 pour éviter que chaque serveur pense être seul et essaie de gérer les services en même temps.

L'*IP flottante*, aussi appelée VIP, est une adresse IP virtuelle qui n'appartient à aucun serveur en particulier mais qui se déplace automatiquement vers le serveur actif. Les utilisateurs se connectent toujours à cette IP sans se soucier de quel serveur répond vraiment.

Une *ressource* est tout ce que Pacemaker peut gérer, comme un service web, une adresse IP ou un disque dur.

Pour gérer chaque type de ressource, Pacemaker utilise des *agents* de ressource qui sont des scripts spécialisés. On peut aussi créer des contraintes pour dire à Pacemaker que certaines ressources doivent tourner sur le même serveur ou démarrer dans un ordre précis.

4. Mise en oeuvre

A présent, nous allons mettre en œuvre ces technologies.

4.1. Configuration des machines

Tout d'abord, il est nécessaire d'installer deux machines Debian: l'une nommée *node1* avec une adresse IP en .10 et l'autre nommée *node2* en .20.

Pour une question de rapidité, il est possible de réaliser les manipulations sur le premier nœud et de le cloner pour adapter la configuration pour le nœud 2.

Voici les étapes à suivre pour configurer les machines:

Etape	Description
1	Changer l'adresse IP et la passerelle des machines avec **nano /etc/network/interfaces**.
	Représentation
	`root@bullseye:~# nano /etc/network/interfaces`

	allow-hotplug enp0s3 iface enp0s3 inet static address 192.168.56.10/24 gateway 192.168.56.254 allow-hotplug enp0s3 iface enp0s3 inet static address 192.168.56.20/24 gateway 192.168.56.254
Etape	Description
2	Changer le nom des machines avec **nano /etc/hostname**.
	Représentation
	root@bullseye:~# nano /etc/hostname node1 node2
Etape	Description
3	Les machines *node1* et *node2* doivent être déclarées dans **/etc/hosts**.
	Représentation
	root@bullseye:~# nano /etc/hosts 127.0.0.1 localhost 127.0.1.1 node1 192.168.56.20 node2 127.0.0.1 localhost 127.0.1.1 node2 192.168.56.10 node1
Etape	Description
4	Redémarrer la machine avec **reboot**.
	Représentation
	root@node1:~# reboot

4.2. Connexion SSH

Afin de faciliter l'utilisation des machines virtuelles, pour par exemple copier-coller des commandes, il est possible d'utiliser une connexion SSH. Cela permet d'avoir accès à la machine virtuelle depuis l'interpréteur de commandes de la machine hôte, peu importe où se trouve la machine virtuelle, à condition qu'elle soit allumée.

Voici les étapes à réaliser depuis les machines virtuelles:

Etape	Commande	Description
1	apt update	Permet de mettre à jour la liste des paquets. **Cette commande est obligatoire avant toute installation.**

	Représentation	
	root@node1:~# apt update Atteint :1 http://deb.debian.org/debian bullseye InRelease Atteint :2 http://deb.debian.org/debian bullseye-updates InRelease	
Etape	Commande	Description
2	apt install ssh	Permet d'installer un serveur SSH sur la machine virtuelle.
	Représentation	
	root@node1:~# apt install ssh	
Etape	Commande	Description
3	reboot	Permet de redémarrer la machine.
	Représentation	
	root@node1:~# reboot	

S'il y a volonté de vérifier que le paquet SSH est bien en cours de fonctionnement, il est nécessaire d'exécuter la commande **systemctl status ssh**:

```
sio@buster:~$ systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2025-01-28 13:45:25 CET; 43min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 327 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
 Main PID: 342 (sshd)
    Tasks: 1 (limit: 1149)
   Memory: 6.2M
   CGroup: /system.slice/ssh.service
           └─342 /usr/sbin/sshd -D
```

Si SSH n'est pas activé, la commande à entrer est **systemctl enable ssh**:

```
root@buster:~# systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor preset: enabled)
   Active: inactive (dead)
     Docs: man:sshd(8)
           man:sshd_config(5)
```

```
root@buster:~# systemctl enable ssh
Synchronizing state of ssh.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable ssh
Created symlink /etc/systemd/system/sshd.service → /lib/systemd/system/ssh.service.
Created symlink /etc/systemd/system/multi-user.target.wants/ssh.service → /lib/systemd/system/ssh.se
rvice.
```

4.3. Installation d'Apache2

Ensuite, il est nécessaire d'installer Apache2 sur les deux machines. Pour cela, il faut se rendre dans le terminal de la machine virtuelle et suivre les instructions suivantes:

Etape	Commande	Description
1	apt update	Permet de mettre à jour la liste des paquets. **Cette commande est obligatoire avant toute installation.**
	Représentation	

	`root@node1:~# apt update`
	`Atteint :1 http://deb.debian.org/debian bullseye InRelease`
	`Atteint :2 http://deb.debian.org/debian bullseye-updates InRelease`	

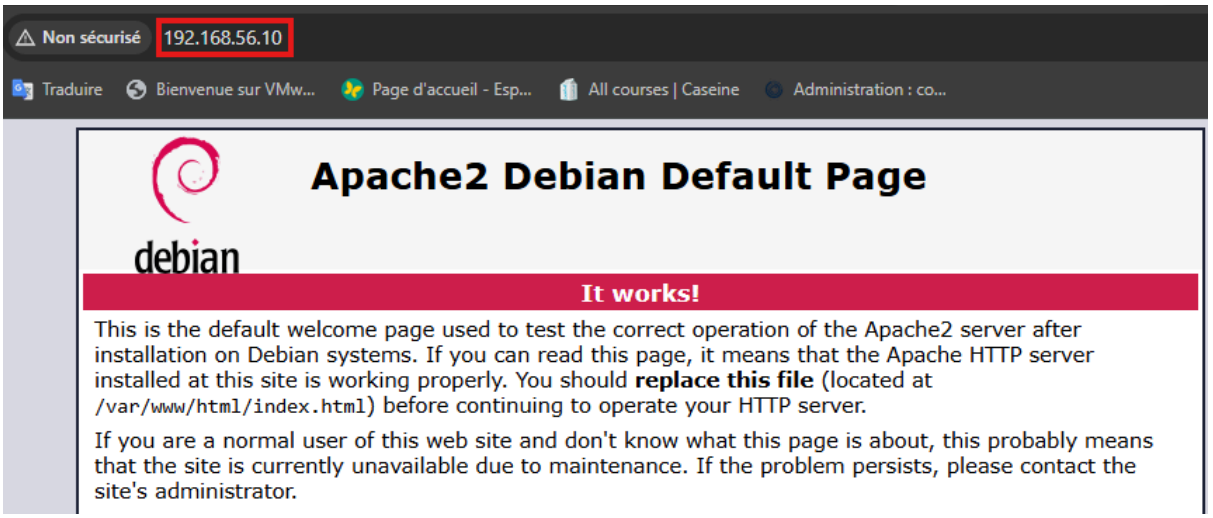
Etape	Commande	Description
2	apt install -y apache2 apache2-doc	Permet d'installer Apache2, la documentation le concernant et de cocher "oui" automatiquement à toutes les questions qui seront posées.
	Représentation	
	`root@node1:~# apt install -y apache2 apache2-doc`	

Pour vérifier que Apache2 est bien en cours de fonctionnement, il est nécessaire d'exécuter la commande **systemctl status apache2**:

```
root@node2:~# systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2025-11-18 11:19:57 CET; 20s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 317 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUCCESS)
   Main PID: 384 (apache2)
      Tasks: 55 (limit: 1115)
     Memory: 11.8M
        CPU: 82ms
     CGroup: /system.slice/apache2.service
             ├─384 /usr/sbin/apache2 -k start
             ├─390 /usr/sbin/apache2 -k start
             └─391 /usr/sbin/apache2 -k start

nov. 18 11:19:56 node2 systemd[1]: Starting The Apache HTTP Server...
nov. 18 11:19:57 node2 apachectl[359]: AH00558: apache2: Could not reliably determine the server's
nov. 18 11:19:57 node2 systemd[1]: Started The Apache HTTP Server.
```

Pour vérifier que tout a été correctement installé, il est nécessaire d'entrer l'adresse IP de la machine virtuelle dans le navigateur de notre machine hôte:



On observe que l'on obtient bien la page par défaut d'Apache2. Cela est à effectuer pour les deux machines Debian.

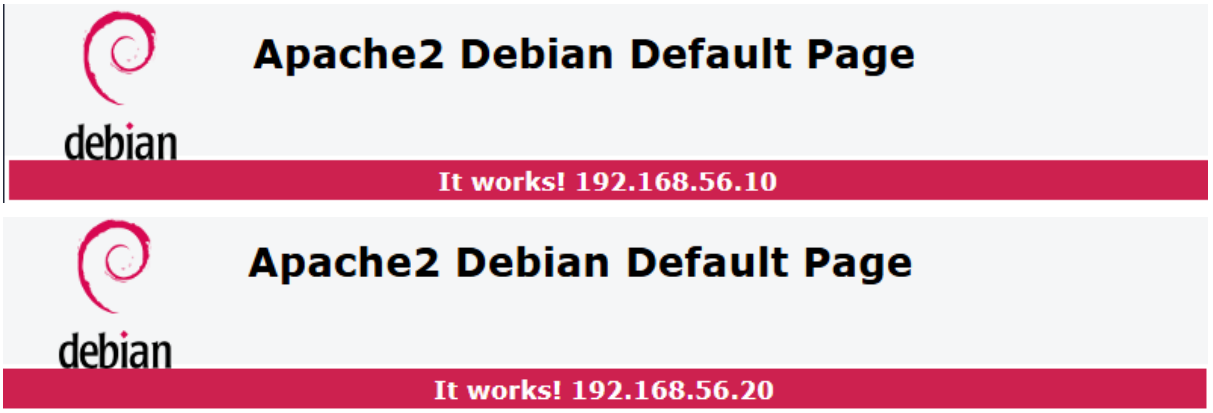
Pour chaque machine, il est important de personnaliser la page par défaut d'Apache pour faire apparaître l'adresse IP de la machine. De cette manière, le failover pourra se voir.

On réalise cela avec la commande **nano /var/www/html/index.html**:

```
<div class="section_header section_header_red">
  <div id="about"></div>
  It works! 192.168.56.10
</div>
```

```
<div class="section_header section_header_red">
  <div id="about"></div>
  It works! 192.168.56.20
</div>
<div class="content_section_text">
```

On vérifie les changements en allant dans le navigateur:



4.4. Installation du cluster

Nous allons maintenant installer et configurer les composants du cluster : Corosync et Pacemaker:

Etape	Description
1	Sur *node1*, installer les paquets Corosync, Pacemaker et PCS (Pacemaker Configuration System) avec **apt install corosync pacemaker pcs -y**.
	Représentation
	`root@node1:~# apt install corosync pacemaker pcs -y`
Etape	Description
2	Répéter l'installation sur *node2*.
	Représentation
	`root@node2:~# apt install corosync pacemaker pcs -y`
Etape	Description
3	Sur *node1*, modifier le fichier de configuration de Corosync avec **nano /etc/corosync/corosync.conf**
	Représentation
	`root@node1:~# nano /etc/corosync/corosync.conf`
Etape	Description
4	Entrer la configuration adaptée dans le fichier.
	Représentation

```
totem {
        version: 2
        cluster_name: cluster-web
        transport: udpu
      }
```

```
nodelist {
         node {
                ring0_addr: 192.168.56.10
                name: node1
                nodeid: 1
              }
         node {
                ring0_addr: 192.168.56.20
                name: node2
                nodeid: 2
              }
         }

quorum {
         provider: corosync_votequorum
         two_node: 1
       }

logging {
          to_logfile: yes
          logfile: /var/log/corosync/corosync.log
          to_syslog: yes
        }
```

Etape	Description
5	Faire la même chose pour le *node2*.
	Représentation

```
totem {
        version: 2
        cluster_name: cluster-web
        transport: udpu
      }
```

```
nodelist {
         node {
                ring0_addr: 192.168.56.10
                name: node1
                nodeid: 1
              }
         node {
                ring0_addr: 192.168.56.20
                name: node2
                nodeid: 2
              }
         }

quorum {
         provider: corosync_votequorum
         two_node: 1
       }

logging {
          to_logfile: yes
          logfile: /var/log/corosync/corosync.log
          to_syslog: yes
        }
```

Etape	Description
6	Vérifier la synchronisation des 2 nœuds avec **timedatectl status**, sur les deux machines.
	Représentation

<table>
<tr><td></td><td>

```
root@node1:~# timedatectl status
               Local time: jeu. 2025-11-20 14:43:30 CET
           Universal time: jeu. 2025-11-20 13:43:30 UTC
                 RTC time: jeu. 2025-11-20 13:43:31
                Time zone: Europe/Paris (CET, +0100)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

```
root@node2:~# timedatectl status
               Local time: jeu. 2025-11-20 14:44:09 CET
           Universal time: jeu. 2025-11-20 13:44:09 UTC
                 RTC time: jeu. 2025-11-20 13:44:10
                Time zone: Europe/Paris (CET, +0100)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

</td></tr>
<tr><td>Etape</td><td>Description</td></tr>
<tr><td rowspan="3">7</td><td>Faire en sorte que le service ne se lance pas automatiquement au démarrage du système avec systemctl disable apache2. Cela sur les deux serveurs.</td></tr>
<tr><td>Représentation</td></tr>
<tr><td>

```
root@node1:~# systemctl disable apache2
root@node2:~# systemctl disable apache2
```

</td></tr>
<tr><td>Etape</td><td>Description</td></tr>
<tr><td rowspan="3">8</td><td>Arrêter le service Apache2 sur les deux serveurs avec systemctl stop apache2.</td></tr>
<tr><td>Représentation</td></tr>
<tr><td>

```
root@node1:~# systemctl stop apache2
root@node2:~# systemctl stop apache2
```

</td></tr>
<tr><td>Etape</td><td>Description</td></tr>
<tr><td rowspan="3">9</td><td>Sur node1, activer le démarrage automatique de Corosync et Pacemaker au boot avec systemctl enable corosync pacemaker.</td></tr>
<tr><td>Représentation</td></tr>
<tr><td>

```
root@node1:~# systemctl enable corosync pacemaker
```

</td></tr>
<tr><td>Etape</td><td>Description</td></tr>
<tr><td>10</td><td>Démarrer Corosync sur node1 avec systemctl start corosync.</td></tr>
</table>

	Représentation
	`root@node1:~# systemctl start corosync`
Etape	Description
11	Démarrer Pacemaker sur *node1* avec **systemctl start pacemaker.** Attention, Pacemaker doit être démarré après Corosync car il dépend de lui.
	Représentation
	`root@node1:~# systemctl start pacemaker`
Etape	Description
12	Répéter les étapes 6 à 8 sur *node2* pour démarrer les services du cluster.
	Représentation
	-
Etape	Description
13	Vérifier l'état du cluster avec la commande **pcs status.**
	Représentation

```
root@node2:~# pcs status
Cluster name: cluster-web

WARNINGS:
No stonith devices and stonith-enabled is not false

Cluster Summary:
  * Stack: corosync
  * Current DC: node1 (version 2.0.5-ba59be7122) - partition with quorum
  * Last updated: Tue Nov 18 11:40:54 2025
  * Last change:  Tue Nov 18 11:24:41 2025 by hacluster via crmd on node1
  * 1 node configured
  * 0 resource instances configured

Node List:
  * Online: [ node1 ]

Full List of Resources:
  * No resources

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

4.5. Configuration du cluster

Maintenant que le cluster est fonctionnel, nous allons le configurer:

Etape	Description
1	Désactiver temporairement STONITH (Shoot The Other Node In The Head) sur *node1* avec **pcs property set stonith-enabled=false**. STONITH est un mécanisme de protection qui éteint un nœud défaillant, mais nécessite du matériel spécial. Dans un environnement de test, on le désactive.
	Représentation
	`root@node1:~# pcs property set stonith-enabled=false`
Etape	Description
2	Désactiver la politique de quorum pour un cluster à deux nœuds avec **pcs property set no-quorum-policy=ignore**
	Représentation
	`root@node1:~# pcs property set no-quorum-policy=ignore`
Etape	Description
3	Créer la ressource IP flottante entre les nœuds, sur *node1* avec **pcs resource create virtual_ip ocf:heartbeat:IPaddr2 ip=192.168.56.100 cidr_netmask=24 op monitor interval=30s**
	Représentation
	`root@node1:~# pcs resource create virtual_ip ocf:heartbeat:IPaddr2 ip=192.168.56.100 cidr_netmask=24 op monitor interval=30s`
Etape	Description
4	Donner l'ordre à Pacemaker de gérer le serveur Web Apache avec **/usr/sbin/pcs resource create WebService ocf:heartbeat:apache \configfile="/etc/apache2/apache2.conf" \op monitor interval=30s**
	Représentation
	`root@node1:/usr/sbin# /usr/sbin/pcs resource create WebService ocf:heartbeat:apache \configfile="/etc/apache2/apache2.conf" \op monitor interval=30s`
Etape	Description
5	Demander à Pacemaker de créer un groupe nommé WebCluster et d'y mettre les 2 ressources: virtual_ip et WebService, avec

	/usr/sbin/pcs resource group add WebCluster virtual_ip WebService.
	Représentation
	`root@node1:/usr/sbin# /usr/sbin/pcs resource group add WebCluster virtual_ip WebService`
Etape	Description
6	Vérifier que la ressource IP virtuelle a bien été créée et est active avec **pcs status**.
	Représentation

	`root@node1:~# pcs status`
	`Cluster name: clusterweb`
	
	`WARNINGS:`
	`No stonith devices and stonith-enabled is not false`
	
	`Cluster Summary:`
	`  * Stack: corosync`
	`  * Current DC: node2 (version 2.0.5-ba59be7122) - partition with quorum`
	`  * Last updated: Thu Nov 20 15:20:26 2025`
	`  * Last change:  Thu Nov 20 15:16:36 2025 by root via cibadmin on node1`
	`  * 2 nodes configured`
	`  * 2 resource instances configured`
	
	`Node List:`
	`  * Online: [ node1 node2 ]`
	
	`Full List of Resources:`
	`  * Resource Group: WebCluster:`
	`    * virtual_ip  (ocf::heartbeat:IPaddr2):  Stopped`
	`    * WebService  (ocf::heartbeat:apache):  Stopped`
	
	`Daemon Status:`
	`  corosync: active/enabled`
	`  pacemaker: active/enabled`
	`  pcsd: active/enabled`

Etape	Description
7	Vérifier le bon fonctionnement de l'ensemble et déceler les éventuelles erreurs avec **systemctl status corosync**.
	Représentation

	root@node1:~# systemctl status corosync ● corosync.service - Corosync Cluster Engine Loaded: loaded (/lib/systemd/system/corosync.service; enabled; vendor preset: enabled) Active: active (running) since Fri 2025-11-21 13:36:48 CET; 5min ago Docs: man:corosync man:corosync.conf man:corosync_overview Main PID: 337 (corosync) Tasks: 9 (limit: 1115) Memory: 154.2M CPU: 6.012s CGroup: /system.slice/corosync.service └─337 /usr/sbin/corosync -f nov. 21 13:36:52 node1 corosync[337]: [KNET] host: host: 2 (passive) best link: 0 (pri: 1) nov. 21 13:36:52 node1 corosync[337]: [KNET] pmtud: Global data MTU changed to: 522 nov. 21 13:36:52 node1 corosync[337]: [QUORUM] Sync members[2]: 1 2 nov. 21 13:36:52 node1 corosync[337]: [QUORUM] Sync joined[1]: 2 nov. 21 13:36:52 node1 corosync[337]: [TOTEM] A new membership (1.1b) was formed. Members joined: 2 nov. 21 13:36:52 node1 corosync[337]: [QUORUM] This node is within the primary component and will provide service. nov. 21 13:36:52 node1 corosync[337]: [QUORUM] Members[2]: 1 2 nov. 21 13:36:52 node1 corosync[337]: [MAIN] Completed service synchronization, ready to provide service. nov. 21 13:37:09 node1 corosync[337]: [KNET] pmtud: PMTUD link change for host: 2 link: 0 from 522 to 1446 nov. 21 13:37:09 node1 corosync[337]: [KNET] pmtud: Global data MTU changed to: 1446
Etape	Description
8	Vérifier que l'IP virtuelle est bien présente sur l'interface réseau du nœud actif avec **ip a.** Représentation root@node1:~# ip a 1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default q len 1000 link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00 inet 127.0.0.1/8 scope host lo valid_lft forever preferred_lft forever inet6 ::1/128 scope host valid_lft forever preferred_lft forever 2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gro up default qlen 1000 link/ether 08:00:27:c8:7a:45 brd ff:ff:ff:ff:ff:ff inet 192.168.56.10/24 brd 192.168.56.255 scope global enp0s3 valid_lft forever preferred_lft forever inet 192.168.56.100/24 brd 192.168.56.255 scope global secondary enp0s3 valid_lft forever preferred_lft forever inet6 fe80::a00:27ff:fec8:7a45/64 scope link valid_lft forever preferred_lft forever

Pour l'étape 3, afin de créer l'adresse IP flottante, il est possible d'entrer la commande **pcs configure primitive VirtualIP ocf:heartbeat:IPaddr2 \params ip=192.168.56.100 cidr_netmask=24 op monitor interval=10s:**

```
root@node1:~# pcs configure primitive VirtualIP ocf:heartbeat:IPaddr2 \params ip=192.168.56.100 cidr_netmask=24 op monit
or interval=10s
```

Simplement il faut que dans toutes les commandes utilisées, le nom des ressources soit le même.

4.6. Vérifications

Suite à nos nombreuses installations et configurations, il est important de procéder à des vérifications.

4.6.1. Vérifier l'installation d'une application

Afin de vérifier qu'une application est bien installée, il suffit d'entrer la commande **dpkg -l** suivie de l'application. Par exemple, il est possible d'essayer avec dpkg -l pcs:

```
root@node1:~# dpkg -l pcs
Souhait=inconnU/Installé/suppRimé/Purgé/H=à garder
| État=Non/Installé/fichier-Config/dépaqUeté/échec-conFig/H=semi-installé/W=attend-traitement-déclenchements
|/ Err?=(aucune)/besoin Réinstallation (État,Err: majuscule=mauvais)
||/ Nom           Version          Architecture Description
+++-==============-================-============-=================================
ii  pcs           0.10.8-1+deb11u1 all          Pacemaker Configuration System
```

Les deux i à gauche du nom de l'application signifient que cette dernière est bien installée.

L'application est bien présente dans le répertoire /usr/sbin:

```
root@node1:~# cd /usr/sbin
root@node1:/usr/sbin# ls
a2disconf               e2undo                fstab-decode              ocf-tester
a2dismod                e4crypt               fstrim                    on_ac_power
a2dissite               e4defrag              genl                      openhpid
a2enconf                ebtables              getcap                    ownership
a2enmod                 ebtables-nft          getpcaps                  pacemakerd
a2ensite                ebtables-nft-restore  getty                     pam-auth-update
a2query                 ebtables-nft-save     groupadd                  pam_getenv
aa-remove-unknown       ebtables-restore      groupdel                  pam_timestamp_check
aa-status               ebtables-save         groupmems                 pcs
aa-teardown             faillock              groupmod                  pcsd
```

4.6.2. Vérification du failover

Afin de vérifier que le failover fonctionne bien, il faut dans un premier temps arrêter Pacemaker sur *node1* avec **systemctl stop pacemaker**:

```
root@node1:~# systemctl stop pacemaker
```

Ensuite, en faisant **ip a** sur *node2*, on observe qu'il a récupéré l'adresse flottante:

```
root@node2:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default q
len 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gro
up default qlen 1000
    link/ether 08:00:27:21:ec:63 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.20/24 brd 192.168.56.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet 192.168.56.100/24 brd 192.168.56.255 scope global secondary enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe21:ec63/64 scope link
       valid_lft forever preferred_lft forever
```

Dans le navigateur si l'on entre l'adresse flottante, on tombe sur la page Apache2 de *node2*:

Pour redémarrer Pacemaker sur *node1* et faire en sorte qu'il récupère cette adresse IP flottante, il suffit d'entrer **systemctl start pacemaker**:

```
root@node1:~# systemctl start pacemaker
```

Puis d'entrer:

```
root@node1:~# pcs resource move virtual_ip node1
```

Ainsi, *node1* récupère l'adresse IP flottante:

```
root@node1:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default q
len 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gro
up default qlen 1000
    link/ether 08:00:27:c8:7a:45 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.10/24 brd 192.168.56.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet 192.168.56.100/24 brd 192.168.56.255 scope global secondary enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fec8:7a45/64 scope link
       valid_lft forever preferred_lft forever
```