# Et PAF, ça bascule!


26 juin 2018

Stefan Fercot

true

# Et PAF, ça bascule!

TITRE : Et PAF, ça bascule!
SOUS-TITRE :

DATE: 26 juin 2018

## QUI SUIS-JE?

- Stefan Fercot
- aka. pgstef
- utilise PostgreSQL depuis 2010
- dans la communauté depuis 2016
- @dalibo depuis 2017

---

Et PAF, ça bascule!

## DALIBO

- Services

| Support | Formation | Conseil |
|---|---|---|

- Participation active à la communauté
- Dalibo recrute des DBA !!!

N'hésitez pas à venir nous rencontrer sur le stand pour en savoir plus !

---

# INTRODUCTION

---

## POSTGRESQL ET LA RÉPLICATION EN FLUX

- système robuste et performant
- sécurité des données
- pas de service hautement disponible

PostgreSQL bénéficie aujourd'hui d'un système de réplication en flux robuste et performant. Bien que cela assure une très bonne sécurité en terme de perte de données, cela ne rend pas le service PostgreSQL hautement disponible pour autant.

---

## BASCULE AUTOMATIQUE - COMPLEXITÉ

- comment détecter une vraie défaillance ?
- le maître ne répond plus ?
- est-il vraiment éteint ?
- est-il vraiment inactif ?
- comment réagir à une panne réseau ?
- comment éviter les split brain ?

---

## PACEMAKER

- est **LA** référence de la HA sous Linux
- est un "Cluster Resource Manager"
- supporte les fencing, quorum et watchdog
- s'interface avec n'importe quel service
- multi-service, avec dépendances, ordre, contraintes, règles, etc
- chaque service a son **Resource Agent**

---

Et PAF, ça bascule!

## ET PAF DANS TOUT ÇA?

- garder Pacemaker: il fait tout et bien à notre place
- se concentrer sur notre métier: PostgreSQL
- PAF est un **Resource Agent** utilisé par Pacemaker

… PAF n'est qu'une brique (RA OCF pour Pacemaker)

---

# INSTALLATION

La documentation fourni plusieurs guides d'installation rapide: http://clusterlabs.github.io/PAF/documenta

---

## PRÉREQUIS

- 2 nœuds ou plus
- PostgreSQL installé
- réplication fonctionnelle
- template recovery.conf.pcmk
  - `standby_mode = on`
  - `primary_conninfo = 'host=ha-vip application_name=hostname'`
  - `recovery_target_timeline = 'latest'`

Pour cette démo, 3 machines virtuelles ont été créées avec qemu-kvm.

Il s'agit de 3 serveurs sous CentOS 7 (hanode1, hanode2, hanode3).

PostgreSQL 10 y a été installé via les paquets du PGDG.

```
# yum install -y https://download.postgresql.org/pub/repos/yum/10/redhat/rhel-7-x86_64
# yum install -y postgresql10-server
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --permanent --add-service=postgresql
# firewall-cmd --reload
```

Création des templates:

```
# cat ~postgres/recovery.conf.pcmk
standby_mode = on
primary_conninfo = 'host=ha-vip application_name=hanode1'
recovery_target_timeline = 'latest'
# chown postgres:postgres ~postgres/recovery.conf.pcmk
# chmod 700 ~postgres/recovery.conf.pcmk
```

Initialisation du primaire:

```
# /usr/pgsql-10/bin/postgresql-10-setup initdb
# echo "listen_addresses = '*'" >> /var/lib/pgsql/10/data/postgresql.conf
# echo "host replication all all trust" >> /var/lib/pgsql/10/data/pg_hba.conf
# systemctl start postgresql-10.service
```

Création des secondaires:



true

Et PAF, ça bascule!

```
# pg_basebackup -h hanode1 -D /var/lib/pgsql/10/data -R
# chown -R postgres:postgres /var/lib/pgsql/10/data/
# systemctl start postgresql-10.service
```

Dès lors que la réplication est fonctionnelle, arrêter proprement les services PostgreSQL:

```
# systemctl stop postgresql-10.service
```

Remarque: il faut également interdire à chaque nœud de se répliquer lui-même.

```
[root@hanode1 ~]# cat /var/lib/pgsql/10/data/pg_hba.conf |grep reject
host replication all hanode1 reject
```

---

## INSTALLER PACEMAKER ET PAF

```
yum install -y pacemaker resource-agents pcs
yum install -y fence-agents-all fence-agents-virsh
yum install -y resource-agents-paf
systemctl disable corosync
systemctl disable pacemaker
```

Le paquet `resource-agents-paf` est disponible directement dans le dépôt PGDG.

Il est recommandé de désactiver Pacemaker et Corosync au démarrage du serveur. Cela permet à l'administrateur de s'assurer de l'état du nœud avant de le réintégrer au cluster suite à un incident.

L'outil `pcs` fonctionne désormais très bien sur les distributions Debian récentes.

---

## AUTHENTIFIER CHAQUE NŒUD À L'AIDE DE L'OUTIL D'ADMINISTRATION PCS

```
systemctl enable pcsd
systemctl start pcsd
passwd hacluster
pcs cluster auth hanode1 hanode2 hanode3 -u hacluster
```

Le daemon `pcsd` s'occupe de propager les configurations et commandes sur tous les nœuds.

L'outil `pcs` se sert de l'utilisateur système `hacluster` pour s'authentifier auprès de pcsd. Puisque les commandes de gestion du cluster peuvent être exécutées depuis n'importe quel membre du cluster, il est recommandé de configurer le même mot de passe pour cet utilisateur sur tous les nœuds pour éviter les confusions.

---

## CRÉER LE CLUSTER

```
pcs cluster setup --name cluster_pgsql hanode1 hanode2 hanode3
pcs cluster start --all
pcs resource defaults migration-threshold=3
pcs resource defaults resource-stickiness=10
```

Changer certaines valeurs par défaut pour les ressources créées plus tard:

- migration-threshold: contrôle combien de fois le cluster tente de "réparer" une ressource sur un même nœud avant de la déplacer sur un autre.
- resource-stickiness: ajoute du poids à la ressource sur son nœud courant pour éviter que celle-ci ne fasse des allées et venues entre les différents nœuds où elle aurait le même poids.

---

## ÉTAT DU CLUSTER - 1

---

## FENCING

- chaque brique doit toujours être dans un état déterminé
- garantie de pouvoir sortir du système un élément défaillant
- implémenté dans Pacemaker au travers du daemon `stonithd`

Lorsqu'un serveur n'est plus accessible au sein d'un cluster, il est impossible aux autres nœuds de déterminer l'état réel de ce dernier:

- a-t-il crashé ?
- est-ce un problème réseau ?
- subit-il une forte charge temporaire ?

Le seul moyen de répondre à ces questions est d'éteindre ou d'isoler le serveur. Cette action permet de déterminer de façon certaine son statut.



true

Et PAF, ça bascule!

Passer outre ce mécanisme, c'est s'exposer de façon certaine à des situations de split brain où deux instances PostgreSQL sont accessibles en écriture au sein du cluster, mais ne répliquent pas entre elles. Réconcilier les données entre ces deux instances peut devenir un véritable calvaire et provoquer une ou plusieurs indisponibilités.

---

## CONFIGURATION DU FENCING

- Une ressource de fencing pour chaque nœud

```
pcs stonith create fence_vm_hanode1 fence_virsh             \
  pcmk_host_check="static-list" pcmk_host_list="hanode1"    \
  ipaddr="192.168.122.1" login="root" port="hanode1"        \
  action="reboot" identity_file="/root/.ssh/id_rsa"
```

- Une contrainte d'exclusion pour chacune de ces ressources

```
pcs constraint location fence_vm_hanode1 avoids hanode1=INFINITY
```

Création d'une ressource de fencing pour chaque nœud. Ici spécifique à la technologie de virtualisation utilisée pour la démo.

Ajout ensuite de contraintes d'exclusion pour que chaque STONITH évite le nœud dont il est responsable.

```
# pcs cluster cib cluster1.xml

# pcs -f cluster1.xml stonith create fence_vm_hanode1 fence_virsh \
    pcmk_host_check="static-list" pcmk_host_list="hanode1"        \
    ipaddr="192.168.122.1" login="root" port="hanode1"            \
    action="reboot" identity_file="/root/.ssh/id_rsa"

# pcs -f cluster1.xml stonith create fence_vm_hanode2 fence_virsh \
    pcmk_host_check="static-list" pcmk_host_list="hanode2"        \
    ipaddr="192.168.122.1" login="root" port="hanode2"            \
    action="reboot" identity_file="/root/.ssh/id_rsa"

# pcs -f cluster1.xml stonith create fence_vm_hanode3 fence_virsh \
    pcmk_host_check="static-list" pcmk_host_list="hanode3"        \
    ipaddr="192.168.122.1" login="root" port="hanode3"            \
    action="reboot" identity_file="/root/.ssh/id_rsa"
```

```
# pcs -f cluster1.xml constraint location fence_vm_hanode1 avoids hanode1=INFINITY
# pcs -f cluster1.xml constraint location fence_vm_hanode2 avoids hanode2=INFINITY
# pcs -f cluster1.xml constraint location fence_vm_hanode3 avoids hanode3=INFINITY
# pcs cluster cib-push cluster1.xml
```

---

## ÉTAT DU CLUSTER - 2

---

## CRÉATION DES RESSOURCES POSTGRESQL - 1

- pgsqld
  - propriétés de l'instance PostgreSQL

```
pcs resource create pgsqld ocf:heartbeat:pgsqlms          \
  bindir=/usr/pgsql-10/bin pgdata=/var/lib/pgsql/10/data  \
  recovery_template=/var/lib/pgsql/recovery.conf.pcmk     \
  op start timeout=60s                                    \
  op stop timeout=60s                                     \
  op promote timeout=30s                                  \
  op demote timeout=120s                                  \
  op monitor interval=15s timeout=10s role="Master"       \
  op monitor interval=16s timeout=10s role="Slave"        \
  op notify timeout=60s
```

PostgreSQL ne disposant pas de demote à chaud, le timeout demote correspond à stop + start.

---



true

Et PAF, ça bascule!

## ÉTAT DU CLUSTER - 3

---

## CRÉATION DES RESSOURCES POSTGRESQL - 2

- `pgsql-ha`
  - clone la ressource `pgsqld` sur l'ensemble des nœuds du cluster
  - décide où est promue l'instance primaire,...

```
pcs resource master pgsql-ha pgsqld notify=true
```

---

## ÉTAT DU CLUSTER - 4

---

## CRÉATION DES RESSOURCES POSTGRESQL - 3

- `pgsql-master-ip`
  - contrôle l'IP virtuelle ha-vip (192.168.122.110)
  - doit être démarrée sur le nœud hébergeant la ressource maître

```
pcs resource create pgsql-master-ip ocf:heartbeat:IPaddr2 \
  ip=192.168.122.110 cidr_netmask=24                      \
  op monitor interval=10s

pcs constraint colocation add pgsql-master-ip \
  with master pgsql-ha INFINITY
```

L'utilisation d'une IP virtuelle n'est pas obligatoire. Il s'agit toutefois de la méthode la plus couramment employée.

D'autres technologies plus ou moins complexes à mettre en œuvre existent pour permettre de rediriger les connexions vers l'instance maître.

---

## AJOUT DE CONTRAINTES D'ORDRE

- l'IP virtuelle doit rester fonctionnelle sur le maître pendant le demote
- l'IP virtuelle ne doit être démarrée sur le nouveau maître qu'après promote

L'ordre de start/stop et promote/demote doit donc être asymétrique.

```
pcs constraint order promote pgsql-ha \
  then start pgsql-master-ip symmetrical=false kind=Mandatory

pcs constraint order demote pgsql-ha \
  then stop pgsql-master-ip symmetrical=false kind=Mandatory
```

---

## ÉTAT DU CLUSTER - 5

---

## VALIDATION - 1

- `pcs status`

```
Cluster name: cluster_pgsql
Stack: corosync
Current DC: hanode2 (version 1.1.16-12.el7_4.8-94ff4df)
                                - partition with quorum
Last updated: ...
Last change: ... by root via crm_attribute on hanode1

3 nodes configured
7 resources configured

Online: [ hanode1 hanode2 hanode3 ]

...
```

---



true

Et PAF, ça bascule!

## VALIDATION - 2

```
...

Full list of resources:

 fence_vm_hanode1 (stonith:fence_virsh):  Started hanode2
 fence_vm_hanode2 (stonith:fence_virsh):  Started hanode1
 fence_vm_hanode3 (stonith:fence_virsh):  Started hanode1
 Master/Slave Set: pgsql-ha [pgsqld]
     Masters: [ hanode1 ]
     Slaves: [ hanode2 hanode3 ]
 pgsql-master-ip  (ocf::heartbeat:IPaddr2): Started hanode1
```

---

## VÉRIFICATION

```
$ psql -h ha-vip
postgres=# SELECT application_name, client_addr, state, sync_state
             FROM pg_stat_replication;
 application_name |   client_addr   |   state   | sync_state
------------------+-----------------+-----------+------------
 hanode3          | 192.168.122.113 | streaming | async
 hanode2          | 192.168.122.112 | streaming | async
(2 rows)

postgres=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 f
(1 row)
```

---

# GESTION DU CLUSTER

---

## AFFICHER LES RESSOURCES

- `pcs resource show`

```
Master/Slave Set: pgsql-ha [pgsqld]
    Masters: [ hanode1 ]
    Slaves: [ hanode2 hanode3 ]
pgsql-master-ip    (ocf::heartbeat:IPaddr2):   Started hanode1
```

---

## STOP

```
pcs resource disable pgsql-ha
pcs cluster stop --all
```

Désactiver d'abord la ressource maître avant d'arrêter le cluster.

---

## START

```
pcs cluster start --all
pcs resource enable pgsql-ha
```

Ordre inverse du stop.

---



true

Et PAF, ça bascule!

## EXCLURE UN NŒUD DU CLUSTER

- Exclure

```
pcs resource ban --wait pgsql-ha hanode2
```

- Réintégrer

```
pcs resource clear pgsql-ha hanode2
```

---

## FORCER LE DÉPLACEMENT D'UNE RESSOURCE

- Déplacer

```
pcs resource move --wait --master pgsql-ha hanode1
```

- Nettoyer les scores

```
pcs resource clear pgsql-ha
```

---

# DÉMO

---

## FAILOVER 1 - DÉMO

- kill -9 des processus postgres

Your browser does not support the video tag.

---

## FAILOVER 1 - EXPLICATIONS

- kill -9 des processus postgres

1. Le monitor de la ressource passe en “failed”
2. Pacemaker redémarre la ressource sur le noeud

---

## FAILOVER 2 - DÉMO

- Suppression du fichier `/var/lib/pgsql/10/data/global/pg_control`

Your browser does not support the video tag.

---

## FAILOVER 2 - EXPLICATIONS

- Suppression du fichier `/var/lib/pgsql/10/data/global/pg_control`

1. L’agent PAF détecte une erreur
2. Monitor en failed
3. Tentative de redémarrage en échec
4. Fencing de hanode1
5. Choix de l’instance avec le LSN le plus élevé
6. Promotion
7. Déplacement de la VIP



true

Et PAF, ça bascule!

PAF détecte une erreur car il vérifie toujours la cohérence entre l'état du cluster et ce qui est écrit dans le fichier pg_control.

L'erreur remontée est ici : "Could not read all datas from controldata file".

---

## FAILOVER 3 - DÉMO

- `echo c > /proc/sysrq-trigger`

Your browser does not support the video tag.

---

## FAILOVER 3 - EXPLICATIONS

- `echo c > /proc/sysrq-trigger`
    1. Détection de l'indisponibilité de hanode1
    2. Fencing de hanode1
    3. Choix de l'instance avec le LSN le plus élevé
    4. Promotion
    5. Déplacement de la VIP

Détection par Corrosync -> réaction plus rapide

---

## SWITCHOVER - DÉMO

Your browser does not support the video tag.

---

## SWITCHOVER - EXPLICATIONS

- hanode2 : demote de `pgsql-ha` et arrêt de `pgsql-master-ip`
- hanode1 : promotion de PostgreSQL
  - Promotion de hanode1
  - Démarrage de la vip sur hanode1

PAF va prendre le temps de s'assurer que le nouveau maître a bien reçu toutes les transactions afin que l'ancien maître puisse être raccroché en nouveau standby.

---

Et PAF, ça bascule!

## CONCLUSION

- RPO (“recovery point objective”): Streaming Replication
- RTO (“recovery time objective”) : bascule automatique

La bascule automatique apporte de la complexité.

En avez-vous vraiment besoin?

---

# OÙ

- site officiel: http://clusterlabs.github.io/PAF/
- code: https://github.com/ClusterLabs/PAF
- rpm ou deb: dans les dépôts PGDG !
- documentation: https://clusterlabs.github.io/PAF/documentation.html
- support: https://github.com/ClusterLabs/PAF/issues
- mailing list : users@clusterlabs.org

---

Et PAF, ça bascule!

## DES QUESTIONS ?