

Workshop Patroni

Haute disponibilité avec Patroni

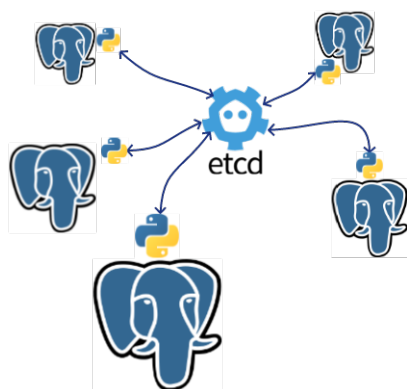


24.02

Contents

1/ Introduction	3
1.1 Objectif de l'atelier	4
1.2 Déroulé de l'atelier	5
1.3 Prérequis de l'atelier	6
2/ Concepts	7
2.1 DCS : etcd	8
2.2 Service PostgreSQL et Patroni	9
3/ Mise en place	11
3.1 Infrastructure	12
3.1.1 Connexion à votre machine virtuelle	12
3.1.2 Playbook Ansible	13
3.2 Installation d'etcd	15
3.2.1 Installation des paquets	15
3.2.1.1 Vérification	15
3.2.2 Configuration du service etcd	16
3.2.3 Démarrage du service	17
3.3 Installation de PostgreSQL / Patroni	19
3.3.1 Configuration de Patroni	20
3.4 Création de l'agrégat	21
3.4.1 Démarrage du primaire	21
3.4.2 Liste des nœuds Patroni	21
3.4.3 Première bascule	21
3.4.3.1 Création de l'utilisateur de réplication	22
3.4.4 Suppression des instances secondaires	22
3.4.5 Démarrage des instances secondaires	23
3.4.6 Vérifications	23
3.4.6.1 Test de bascule manuelle vers chaque nœud	24
3.5 SUPERUSER dédié	26
3.5.1 Après la configuration du premier nœud	26
3.5.2 Avant la configuration du premier nœud	26
4/ Création d'incidents	29
4.1 Perte totale du DCS	30

4.2	Perte du nœud primaire Patroni	32
5/	Modification de la configuration	35
6/	Sauvegardes	39
6.1	Configuration Serveur de pgBackRest	40
6.2	Configuration Client de pgBackRest	41
6.2.1	Création d'une stanza	41
6.2.2	Configuration de l'archivage	41
6.3	Test d'une sauvegarde	43
6.3.1	Détermination de l'instance primaire	43
6.3.1.1	Déclenchement de la sauvegarde	43
7/	Références	45
Notes		47
Notes		49
Notes		51
Nos autres publications		53
Formations		54
Livres blancs		55
Téléchargement gratuit		56
8/	DALIBO, L'Expertise PostgreSQL	57



1/ Introduction

1.1 OBJECTIF DE L'ATELIER

- Mettre en œuvre un cluster Patroni
 - Gérer des bascules
-

1.2 DÉROULÉ DE L'ATELIER

- Durée : 3 heures
 - Concepts
 - Travaux pratiques
 - Mise en place
 - Installation et configuration des services
 - Construction d'un agrégat à bascule automatique
 - Création d'incidents
-

1.3 PRÉREQUIS DE L'ATELIER

- Un terminal
- Une VM Debian Bookworm ou équivalente
- Compétences Linux et PostgreSQL

2/ Concepts

- Arbitrage par un quorum : *DCS etcd*
 - Service PostgreSQL : désactivé
 - Contrôle complet par Patroni
-

2.1 DCS : ETCD

- Arbitre en cas de bascules
- Stockage distribué de la configuration
- Jeton *leader* (etcd)
- Instance primaire PostgreSQL

Pour arbitrer les bascules automatiques, confirmer le primaire PostgreSQL ou distribuer la configuration, Patroni utilise un *DCS* (*distributed configuration system*).

Pour ce rôle, nous utiliserons etcd.

2.2 SERVICE POSTGRESQL ET PATRONI

- Service PostgreSQL à désactiver

Le service PostgreSQL doit être **désactivé** pour ne pas se lancer au démarrage, le contrôle total de l'instance est délégué à Patroni :

```
dalibo@vm:~$
```

```
for node in pg1 pg2 pg3; do
    sudo ssh ${node} "systemctl disable --now postgresql"
    sudo ssh ${node} "systemctl status postgresql"
done
```

```
Synchronizing state of postgresql.service with SysV service script with /lib/systemd/systemd-sysv-install.
```

```
Executing: /lib/systemd/systemd-sysv-install disable postgresql
```

```
Removed "/etc/systemd/system/multi-user.target.wants/postgresql.service".
```

```
○ postgresql.service - PostgreSQL RDBMS
```

```
Loaded: loaded (/lib/systemd/system/postgresql.service; disabled; preset: enabled)
```

```
Active: inactive (dead)
```

```
Jan 30 17:24:42 pg1 systemd[1]: Starting postgresql.service - PostgreSQL RDBMS...
```

```
Jan 30 17:24:42 pg1 systemd[1]: Finished postgresql.service - PostgreSQL RDBMS.
```

```
Jan 31 11:05:23 pg1 systemd[1]: postgresql.service: Deactivated successfully.
```

```
Jan 31 11:05:23 pg1 systemd[1]: Stopped postgresql.service - PostgreSQL RDBMS.
```

```
Synchronizing state of postgresql.service with SysV service script with /lib/systemd/systemd-sysv-install.
```

```
Executing: /lib/systemd/systemd-sysv-install disable postgresql
```

```
Removed "/etc/systemd/system/multi-user.target.wants/postgresql.service".
```

```
○ postgresql.service - PostgreSQL RDBMS
```

```
Loaded: loaded (/lib/systemd/system/postgresql.service; disabled; preset: enabled)
```

```
Active: inactive (dead)
```

```
Jan 30 17:24:44 pg2 systemd[1]: Starting postgresql.service - PostgreSQL RDBMS...
```

```
Jan 30 17:24:44 pg2 systemd[1]: Finished postgresql.service - PostgreSQL RDBMS.
```

```
Jan 31 11:05:25 pg2 systemd[1]: postgresql.service: Deactivated successfully.
```

```
Jan 31 11:05:25 pg2 systemd[1]: Stopped postgresql.service - PostgreSQL RDBMS.
```

```
Synchronizing state of postgresql.service with SysV service script with /lib/systemd/systemd-sysv-install.
```

```
Executing: /lib/systemd/systemd-sysv-install disable postgresql
```

```
Removed "/etc/systemd/system/multi-user.target.wants/postgresql.service".
```

```
○ postgresql.service - PostgreSQL RDBMS
```

```
Loaded: loaded (/lib/systemd/system/postgresql.service; disabled; preset: enabled)
```

```
Active: inactive (dead)
```

```
[...]
```

```
Jan 31 11:05:26 pg3 systemd[1]: Stopped postgresql.service - PostgreSQL RDBMS.
```


3/ Mise en place

3.1 INFRASTRUCTURE

- Connexion à la VM
 - 7 Conteneurs
-

Vous disposez d'une machine virtuelle dédiée dans laquelle nous avons construit 7 conteneurs *LXC* :

- 3 nœuds etcd
 - 3 nœuds Patroni
 - 1 nœud pgBackRest
-

3.1.1 Connexion à votre machine virtuelle

- un point d'entrée commun (> tableau)
 - un port individuellement attribué : 22XX
 - un utilisateur linux : `dalibo`
 - un mot de passe (> tableau)
 - `ssh -p 22XX dalibo@<IP_COMMUNE>`
-

Exemple de configuration `ssh` pour une connexion simplifiée :

```
# ~/.ssh/config
Host vm
Hostname 5.4.3.2 # <IP_COMMUNE>
User dalibo
port 2201 # <PORT_INDIVIDUEL>
```

Test de connexion `ssh` :

```
$ ssh vm
```

```
Last login: Wed Nov 10 13:23:26 2021 from 42.42.42.42
dalibo@vm:~$
```


3.1.2 Playbook Ansible

Infrastructure clef en main par *playbooks Ansible* à cette adresse :

https://public.dalibo.com/exports/formation/workshops/fr/patroni_2024/playbook/etcd-patroni

3 fichiers YAML principaux :

- `inventory.yml` : inventaire des machines
 - `setup.2.yml` : playbook principal
 - `teardown.yml` : playbook de destruction
-

L'infrastructure complète est déjà créée.



La commande pour recréer l'infrastructure sur votre VM est :

```
ansible-playbook -i inventory.yml setup.2.yml
```

La création des conteneurs a été faite ainsi :

```
root@vm:~# ansible-playbook -i inventory.yml -f 7 setup.2.yml
```

```
dalibo@vm:~$ sudo lxc-ls -f
```

NAME	STATE	AUTOSTART	GROUPS	IPV4	IPV6	UNPRIVILEGED
backup	RUNNING	0	-	10.0.3.204	-	false
e1	RUNNING	0	-	10.0.3.101	-	false
e2	RUNNING	0	-	10.0.3.102	-	false
e3	RUNNING	0	-	10.0.3.103	-	false
pg1	RUNNING	0	-	10.0.3.201	-	false
pg2	RUNNING	0	-	10.0.3.202	-	false
pg3	RUNNING	0	-	10.0.3.203	-	false

Sur tous les conteneurs, le fichier `/etc/hosts` est automatiquement renseigné par le *playbook* et devrait contenir au moins :

```
10.0.3.101 e1
10.0.3.102 e2
10.0.3.103 e3
10.0.3.201 pg1
10.0.3.202 pg2
10.0.3.203 pg3
10.0.3.204 backup
```

3.2 INSTALLATION D'ETCD

- Installation des paquets
 - Configuration
 - Démarrage du service
 - Vérification
-

3.2.1 Installation des paquets

- etcd-server
 - iputils-ping
 - vim
-

```
dalibo@vm:~$  
for node in e1 e2 e3; do  
    echo "${node} :"  
    sudo ssh ${node} "apt-get install -y etcd-server iputils-ping"  
done
```

Le démarrage du service est automatique sous Debian.

```
dalibo@vm:~$  
for node in e1 e2 e3; do  
    echo -n "${node} :"  
    sudo ssh ${node} "systemctl is-active etcd"  
done
```

```
e1 : active  
e2 : active  
e3 : active
```

3.2.1.1 Vérification

```
dalibo@vm:~$  
for node in e1 e2 e3; do  
    echo -n "${node} :"  
    sudo ssh ${node} etcdctl member list  
done
```

```
e1 :8e9e05c52164694d, started, e1, http://localhost:2380, http://localhost:2379, false
e2 :8e9e05c52164694d, started, e2, http://localhost:2380, http://localhost:2379, false
e3 :8e9e05c52164694d, started, e3, http://localhost:2380, http://localhost:2379, false
```

Les nœuds sont tous indépendants, ce qui ne nous intéresse pas. Il faut donc les configurer pour qu'ils fonctionnent en agrégat.

Nous arrêtons donc les services :

```
dalibo@vm:~$
for node in e1 e2 e3; do
    echo -n "${node} : "
    sudo ssh ${node} "systemctl stop etcd"
    sudo ssh ${node} "systemctl is-active etcd"
done

e1 : inactive
e2 : inactive
e3 : inactive
```

3.2.2 Configuration du service etcd

- Fichier : `/etc/default/etcd`

La configuration du service etcd se trouve dans le fichier `/etc/default/etcd`, elle doit décrire notre agrégat sur chaque nœud :

- spécifique décrivant le nœud
- une partie commune à tous les nœuds décrivant l'agrégat



Attention aux caractères invisibles ou aux sauts de ligne.

Sur le nœud e1 :

```
# /etc/default/etcd

ETCD_NAME='e1'
ETCD_LISTEN_PEER_URLS='http://127.0.0.1:2380,http://10.0.3.101:2380'
ETCD_LISTEN_CLIENT_URLS='http://127.0.0.1:2379,http://10.0.3.101:2379'
ETCD_INITIAL_ADVERTISE_PEER_URLS='http://10.0.3.101:2380'
ETCD_ADVERTISE_CLIENT_URLS='http://10.0.3.101:2379'

ETCD_INITIAL_CLUSTER_STATE='new'
ETCD_DATA_DIR='/var/lib/etcd/default'
ETCD_INITIAL_CLUSTER_TOKEN='etcd-cluster'
ETCD_INITIAL_CLUSTER='e1=http://10.0.3.101:2380,e2=http://10.0.3.102:2380,e3=http://10.0.3.103:2380'
```

Sur le nœud e2 :

```
# /etc/default/etcd

ETCD_NAME='e2'
ETCD_LISTEN_PEER_URLS='http://127.0.0.1:2380,http://10.0.3.102:2380'
ETCD_LISTEN_CLIENT_URLS='http://127.0.0.1:2379,http://10.0.3.102:2379'
ETCD_INITIAL_ADVERTISE_PEER_URLS='http://10.0.3.102:2380'
ETCD_ADVERTISE_CLIENT_URLS='http://10.0.3.102:2379'

ETCD_INITIAL_CLUSTER_STATE='new'
ETCD_DATA_DIR='/var/lib/etcd/default'
ETCD_INITIAL_CLUSTER_TOKEN='etcd-cluster'
ETCD_INITIAL_CLUSTER='e1=http://10.0.3.101:2380,e2=http://10.0.3.102:2380,e3=http://10.0.3.103:2380'
```

Sur le nœud e3 :

```
# /etc/default/etcd

ETCD_NAME='e3'
ETCD_LISTEN_PEER_URLS='http://127.0.0.1:2380,http://10.0.3.103:2380'
ETCD_LISTEN_CLIENT_URLS='http://127.0.0.1:2379,http://10.0.3.103:2379'
ETCD_INITIAL_ADVERTISE_PEER_URLS='http://10.0.3.103:2380'
ETCD_ADVERTISE_CLIENT_URLS='http://10.0.3.103:2379'

ETCD_INITIAL_CLUSTER_STATE='new'
ETCD_DATA_DIR='/var/lib/etcd/default'
ETCD_INITIAL_CLUSTER_TOKEN='etcd-cluster'
ETCD_INITIAL_CLUSTER='e1=http://10.0.3.101:2380,e2=http://10.0.3.102:2380,e3=http://10.0.3.103:2380'
```

3.2.3 Démarrage du service

- Réinitialisation des bases etcd
- Démarrage du service etcd
- `systemctl start etcd`

Avant de démarrer le service sur chaque nœud, il faut réinitialiser les répertoires de données des nœuds, afin qu'ils repartent sur un répertoire neuf.

Le nœud e1, que nous considérons comme premier *leader* sera démarré en premier :

```
dalibo@vm:~$
for node in e1 e2 e3; do
    echo "${node} :"
    sudo ssh ${node} "rm -vrf ~etcd/default/member"
done
```

```
dalibo@vm:~$  
for node in e1 e2 e3; do  
    echo "${node} :"  
    sudo ssh ${node} "systemctl start etcd" & sleep 1  
done
```

En cas d'échec de démarrage, utilisez la commande *systemd* pour diagnostiquer la cause :

```
root@e1:~# journalctl -xfu etcd
```

Vérification que le nœud e1 ayant démarré en premier, est bien le *leader* :

```
dalibo@vm:~$ sudo ssh e1 "etcdctl endpoint status -w table --cluster"
```

```
+-----+-----+-----+-----+-----+-----+  
|      ENDPOINT      |      ID      | VERSION | DB SIZE | IS LEADER | IS LEARNER |  
+-----+-----+-----+-----+-----+-----+  
| http://10.0.3.101:2379 | 736293150f1cffb7 | 3.4.23 | 41 kB |      true |      false |  
| http://10.0.3.103:2379 | 7ef9d5bb55cefbcc | 3.4.23 | 41 kB |      false |      false |  
| http://10.0.3.102:2379 | 97463691c7858a7b | 3.4.23 | 41 kB |      false |      false |  
+-----+-----+-----+-----+-----+-----+
```

```
/ +-----+-----+-----+-----+  
/ |  RAFT TERM | RAFT INDEX | RAFT APPLIED INDEX | ERRORS |  
/ +-----+-----+-----+-----+  
/ |          2 |          52 |                52 |      |  
/ |          2 |          52 |                52 |      |  
/ |          2 |          52 |                52 |      |  
/ +-----+-----+-----+-----+
```

3.3 INSTALLATION DE POSTGRESQL / PATRONI

- Installation
 - PostgreSQL
 - Patroni
 - pgBackRest

Le dépôt *pgdg* est déjà pré-configuré dans les conteneurs pg1, pg2 et pg3, l'installation est donc triviale :

```
dalibo@vm:~$
for node in pg1 pg2 pg3; do
    echo "${node} :"
    sudo ssh ${node} "apt-get update && apt-get install -y postgresql patroni
    ↪ pgbackrest"
done
```

Vérification :

```
dalibo@vm:~$
for node in pg1 pg2 pg3; do
    echo "${node} :"
    sudo ssh ${node} "dpkg -l postgresql patroni pgbackrest | grep ^ii | cut -d ' ' -f
    ↪ 1,3"
done

pg1 :
ii patroni
ii pgbackrest
ii postgresql
pg2 :
ii patroni
ii pgbackrest
ii postgresql
pg3 :
ii patroni
ii pgbackrest
ii postgresql
```

Le service PostgreSQL doit être désactivé car la gestion totale de l'instance sera déléguée à Patroni :

```
dalibo@vm:~$
for node in pg1 pg2 pg3; do
    echo -n "${node} :"
    sudo ssh ${node} "systemctl is-active postgresql@16-main"
    sudo ssh ${node} "systemctl disable --now postgresql@16-main"
done
```

3.3.1 Configuration de Patroni

Sur tous les nœuds

- Configuration du DCS
 - `/etc/patroni/dcs.yml`
- Génération de la configuration
 - `pg_createconfig_patroni 16 main`

La configuration sous Debian se fait d'abord en renseignant comment contacter le DCS, puis en lançant le script de génération automatique de la configuration de Patroni.

Le fichier est à modifier sur chaque nœud :

```
# /etc/patroni/dcs.yml
etcd3:
  hosts:
    - 10.0.3.101:2379
    - 10.0.3.102:2379
    - 10.0.3.103:2379
```

Une fois le fichier modifié, la configuration peut être générée pour chacun des nœuds :

```
dalibo@vm:~$
for node in pg1 pg2 pg3; do
  echo -n "${node} : "
  sudo ssh ${node} "pg_createconfig_patroni 16 main"
done
```

La configuration `/etc/patroni/16-main.yml` est générée.

3.4 CRÉATION DE L'AGRÉGAT

- Démarrage du primaire
- Création de l'utilisateur de réplication
- Suppression des instances secondaires
- Démarrage des instances secondaires

3.4.1 Démarrage du primaire

La création de l'agrégat commence par la mise en route du primaire sur le nœud pg1, c'est lui qui sera la référence pour les secondaires.

```
dalibo@vm:~$ sudo ssh pg1 "systemctl enable --now patroni@16-main"
```

3.4.2 Liste des nœuds Patroni

Sur chaque nœud Patroni, modifier le fichier `.profile` de l'utilisateur postgres en ajoutant :

```
export PATRONICTL_CONFIG_FILE=/etc/patroni/16-main.yml
```

Lister les nœuds :

```
postgres@pg1:~$ patronictl list
```

```
Current cluster topology
+ Cluster: 16-main (7330941500757972192) -+-----+-----+
| Member | Host          | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1     | 10.0.3.201    | Replica | running |    |          0 |
+-----+-----+-----+-----+-----+-----+
```

3.4.3 Première bascule

Au premier démarrage de Patroni, nous constatons que le nœud pg1 est en lecture seule, il attend une promotion initiale manuelle qui fixera l'état de notre primaire.

```
postgres@pg1:~$ patronictl failover --candidate pg1 --force
```

```
Current cluster topology
+ Cluster: 16-main (7330941500757972192) -+-----+-----+
| Member | Host          | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1     | 10.0.3.201    | Replica | running |    |          0 |
+-----+-----+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+-----+-----+
2024-02-02 15:06:39.21288 Successfully failed over to "pg1"
+ Cluster: 16-main (7330941500757972192) +-----+-----+
| Member | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1     | 10.0.3.201    | Leader | running |    |           |
+-----+-----+-----+-----+-----+-----+
```

Après quelques secondes, la promotion est terminée, la *time line* est renseignée.

```
postgres@pg1:~$ patronictl list
```

```
+ Cluster: 16-main (7330941500757972192) +-----+-----+
| Member | Host          | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1     | 10.0.3.201    | Leader | running | 2  |           |
+-----+-----+-----+-----+-----+-----+
```



En partant d'une instance déjà peuplée, les spécificités de la section *bootstrap* de la configuration de Patroni n'ont pas été appliquées.

```
postgres@pg1:~$ psql
psql (16.1 (Debian 16.1-1.pgdg120+1))
Type "help" for help.
```

```
postgres=# \du
```

```

                        List of roles
Role name |                               Attributes
+-----+-----+
postgres | Superuser, Create role, Create DB, Replication, Bypass RLS
```

L'utilisateur permettant la mise en réplication doit être créé sur ce nœud, avec le mot de passe renseigné dans la configuration de Patroni.

3.4.3.1 Création de l'utilisateur de réplication

```
$ sudo ssh pg1 "sudo -iu postgres psql -c \"CREATE ROLE replicator LOGIN REPLICATION
↳ PASSWORD 'rep-pass'\"" "
```

3.4.4 Suppression des instances secondaires

Les instances secondaires ont été initialisées lors de l'installation du paquet Debian, il faut donc vider leur répertoire de données :

pg1 étant notre primaire :

```
dalibo@vm:~$  
for node in pg2 pg3; do  
    echo "${node} :"  
    sudo ssh ${node} "rm -rf /var/lib/postgresql/16/main/*"  
done
```

Les secondaires seront recréés automatiquement depuis le primaire par Patroni.

3.4.5 Démarrage des instances secondaires

Nous pouvons raccrocher nos secondaires en démarrant les deux instances :

```
dalibo@vm:~$  
for node in pg2 pg3; do  
    echo "${node} :"  
    sudo ssh ${node} "systemctl enable --now patroni@16-main"  
done
```

```
postgres@pg1:~$ patronictl list
```

+ Cluster: 16-main (7330941500757972192) -----+-----+						
Member	Host	Role	State	TL	Lag in MB	
+-----+-----+-----+-----+-----+-----+						
pg1	10.0.3.201	Leader	running	2		
pg2	10.0.3.202	Replica	creating replica		unknown	
pg3	10.0.3.203	Replica	creating replica		unknown	
+-----+-----+-----+-----+-----+-----+						

Après quelques secondes les secondaires sont reconstruits :

```
postgres@pg1:~$ patronictl list
```

+ Cluster: 16-main (7330941500757972192) ---+-----+						
Member	Host	Role	State	TL	Lag in MB	
+-----+-----+-----+-----+-----+-----+						
pg1	10.0.3.201	Leader	running	2		
pg2	10.0.3.202	Replica	streaming	2	0	
pg3	10.0.3.203	Replica	streaming	2	0	
+-----+-----+-----+-----+-----+-----+						

3.4.6 Vérifications

- Test de bascule manuelle vers chaque nœud

3.4.6.1 Test de bascule manuelle vers chaque nœud

```
dalibo@vm:~$ sudo ssh pg1 "sudo -iu postgres patronictl switchover"
```

```
Master [pg1]:
```

```
Candidate ['pg2', 'pg3'] []: pg2
```

```
When should the switchover take place (e.g. 2024-01-31T16:31 ) [now]:
```

```
Current cluster topology
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+
| Member | Host          | Role    | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201   | Leader  | running | 1 |           |
| pg2    | 10.0.3.202   | Replica | running | 1 |           0 |
| pg3    | 10.0.3.203   | Replica | running | 1 |           0 |
+-----+-----+-----+-----+-----+-----+

```

```
Are you sure you want to switchover cluster 16-main, demoting current master
```

```
pg1? [y/N]: y
```

```
2024-01-31 16:31:25.07084 Successfully switched over to "pg2"
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Replica | stopped |    | unknown    |
| pg2    | 10.0.3.202 | Leader  | running | 2  |            |
| pg3    | 10.0.3.203 | Replica | running | 2  | 0          |
+-----+-----+-----+-----+-----+-----+
```

3.5 SUPERUSER DÉDIÉ

Par défaut, le superuser utilisé par Patroni est postgres. Il est possible de remplacer cet utilisateur.

3.5.1 Après la configuration du premier nœud

Modifier le pg_hba.conf dans la configuration distribuée

Dans le fichier /etc/patroni/16-main.yml :

```
postgresql:
  parameters: null
  pg_hba:
    - local    all             postgres                                peer
    - local    all             dba                                  md5
    - host     all             all          10.0.3.201/24          md5
```

- On garde les accès de l'utilisateur postgres en local, authentifié par le système
- On ajoute un accès local à l'utilisateur dba, authentifié par mot de passe

Ajouter l'utilisateur sur le primaire :

```
postgres@pg1:~$ psql
postgres=# CREATE ROLE dba SUPERUSER PASSWORD 'bar';
CREATE ROLE
```

Renseigner le nouveau superuser dans la configuration statique de tous les nœuds :

Dans le fichier /etc/patroni/config.yml.in :

```
[...]
superuser:
  username: "dba"
  password: "bar"
[...]
```

Redémarrer les nœuds :

```
postgres@pg1:~$ patronictl restart 16-main --force
```

3.5.2 Avant la configuration du premier nœud

Modifier le pg_hba.conf dans le template de la configuration statique :

Dans le fichier /etc/patroni/config.yml.in :

```
postgresql:
pg_hba:
- local    all                postgres                                peer
- local    all                dba                                    md5
- host     all                all                @NETWORK@          md5
```

Demander la création de l'utilisateur additionnel dba :

Dans le fichier `/etc/patroni/config.yml.in`:

```
# # Additional users to be created after initializing the cluster
users:
  dba:
    password: bar
    options:
      - superuser
```

Générer la configuration :

```
root@pg-1:~# pg_createconfig_patroni 16 main
```

Redémarrer tous les nœuds :

```
postgres@pg1:~$ patronictl restart 16-main --force
```

Ajouter le nouveau superuser dans la configuration statique de chaque nœud :

Dans le fichier `/etc/patroni/16-main.yml`:

```
[...]
superuser:
  username: "dba"
  password: "bar"
[...]
```

Redémarrer à nouveau les nœuds :

```
postgres@pg1:~$ patronictl restart 16-main --force
```


4/ Création d'incidents

- Perte totale du DCS
 - Freeze du nœud primaire Patroni
 - Bascule manuelle
-

4.1 PERTE TOTALE DU DCS

- Perte de tous les nœuds etcd

Nous simulons un incident majeur au niveau du DCS :

```
dalibo@vm:~$  
for node in e1 e2 e3; do  
    echo "${node} :"  
    sudo lxc-freeze ${node}  
done
```

La commande classique `patronictl list` échoue faute de DCS pour la renseigner.

Nous interrogeons directement les instances :

```
dalibo@vm:~$  
for node in pg1 pg2 pg3; do  
    echo "${node} :"  
    sudo ssh ${node} "sudo -iu postgres psql -c 'SELECT pg_is_in_recovery()'"  
done  
  
pg1 :  
pg_is_in_recovery  
-----  
t  
(1 ligne)  
  
pg2 :  
pg_is_in_recovery  
-----  
t  
(1 ligne)  
  
pg3 :  
pg_is_in_recovery  
-----  
t  
(1 ligne)
```

Nous constatons que l'intégralité des nœuds est passée en lecture seule.

Nous débloquons la situation :

```
dalibo@vm:~$  
for node in e1 e2 e3; do  
    echo "${node} :"  
    sudo lxc-unfreeze ${node}  
done
```

Nous pouvons observer le retour à la normale :

```
postgres@pg1:~$ patronictl list -ew 1
```

4.2 PERTE DU NŒUD PRIMAIRE PATRONI

Perte du primaire courant

Dans un autre terminal, nous observons l'état de l'agrégat sur le nœud pg2 :

```
postgres@pg2:~$ patronictl list -ew 1
```

Nous simulons une perte du primaire pg1 :

```
dalibo@vm:~$ sudo lxc-freeze pg1
```

Nous observons la disparition de pg1 de la liste des nœuds et une bascule automatique se déclenche vers un des nœuds secondaires disponibles :

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg2    | 10.0.3.202 | Replica | running | 7 |          0 |
| pg3    | 10.0.3.203 | Leader  | running | 7 |          |
+-----+-----+-----+-----+-----+-----+
```

Nous rétablissons la situation :

```
dalibo@vm:~$ sudo lxc-unfreeze pg1
```

```
dalibo@vm:~$ sudo ssh pg1 "sudo -iu postgres patronictl list"
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Replica | running | 6 |          0 |
| pg2    | 10.0.3.202 | Replica | running | 7 |          0 |
| pg3    | 10.0.3.203 | Leader  | running | 7 |          |
+-----+-----+-----+-----+-----+-----+
```

Pour un retour à l'état nominal, il suffit de procéder à une bascule manuelle (adapter la commande si votre primaire n'est pas pg3) :

```
postgres@pg1:~$ patronictl switchover --master pg3 --candidate pg1 --force
```

Current cluster topology

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Replica | running | 7 |          0 |
| pg2    | 10.0.3.202 | Replica | running | 7 |          0 |
| pg3    | 10.0.3.203 | Leader  | running | 7 |          |
+-----+-----+-----+-----+-----+-----+
```

```
2021-11-12 13:18:36.05884 Successfully switched over to "pg1"
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Leader | running | 7 |          |
| pg2    | 10.0.3.202 | Replica | running | 7 |          0 |
| pg3    | 10.0.3.203 | Replica | stopped |   |        unknown |
+-----+-----+-----+-----+-----+-----+-----+
```


5/ Modification de la configuration

```
- patronictl edit-config
```

L'un des avantages de bénéficier d'une configuration distribuée est qu'il est possible de modifier cette configuration pour tous les nœuds en une seule opération.

Si le paramètre nécessite un rechargement de la configuration, elle sera lancée sur chaque nœud.

Si la modification nécessite un redémarrage, le drapeau *pending restart* sera positionné sur toutes les instances, qui attendrons une action de votre part pour l'effectuer.

L'installation de la commande `less` est un pré-requis :

```
dalibo@vm:~$  
for node in pg1 pg2 pg3; do  
    sudo ssh ${node} "apt-get install -y less"  
done
```

La modification de la configuration Patroni peut se faire sur n'importe quel nœud :

```
postgres@pg2:~$ patronictl edit-config
```

Nous allons ajouter le paramètre `max_connections`, avec une valeur différente de celle par défaut. Pour ce faire, il faut ajouter une entrée au dictionnaire `parameters` qui a pour dictionnaire parent `postgresql` :

```
loop_wait: 10  
maximum_lag_on_failover: 1048576  
postgresql:  
    parameters:  
        max_connections: 123  
[...]
```

Une confirmation est demandée après avoir quitté l'éditeur :

```
postgres@pg2:~$ patronictl edit-config
```

```
---  
+++  
@@ -1,7 +1,8 @@  
    loop_wait: 10  
    maximum_lag_on_failover: 1048576  
    postgresql:  
-   parameters: null  
+   parameters:  
+       max_connections: 123
```

```
pg_hba:
- local  all          all                                peer
- host   all          all                                127.0.0.1/32    md5
```

Apply these changes? [y/N]: y

Configuration changed

Après modification, il convient de regarder si la modification nécessite ou pas un redémarrage :

```
postgres@pg2:~$ patronictl list -e
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB | Pending restart | Tags |
+-----+-----+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Leader | running | 8 |          | *              |      |
| pg2    | 10.0.3.202 | Replica | running | 8 | 0        | *              |      |
| pg3    | 10.0.3.203 | Replica | running | 8 | 0        | *              |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Dans notre cas, un redémarrage de toutes les instances est nécessaire :

```
postgres@pg2:~$ patronictl restart 16-main
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB | Pending restart |
+-----+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Leader | running | 8 |          | *              |
| pg2    | 10.0.3.202 | Replica | running | 8 | 0        | *              |
| pg3    | 10.0.3.203 | Replica | running | 8 | 0        | *              |
+-----+-----+-----+-----+-----+-----+-----+
```

When should the restart take place (e.g. 2024-01-31T16:37) [now]:

Are you sure you want to restart members pg3, pg2, pg1? [y/N]: y

Restart if the PostgreSQL version is less than provided (e.g. 9.5.2) []:

Success: restart on member pg3

Success: restart on member pg2

Success: restart on member pg1

Vérification du statut Pending restart des nœuds membres de notre cluster :

```
postgres@pg2:~$ patronictl list -e
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+-----+-----+
| Member | Host      | Role   | State  | TL | Lag in MB | Pending restart | Tags |
+-----+-----+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Leader | running | 8 |          |                |      |
| pg2    | 10.0.3.202 | Replica | running | 8 | 0        |                |      |
| pg3    | 10.0.3.203 | Replica | running | 8 | 0        |                |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
```


Nous pouvons vérifier que la nouvelle valeur du paramètre `max_connections` est bien prise en compte sur l'ensemble des instances PostgreSQL :

```
dalibo@vm:~$
for node in pg1 pg2 pg3; do
    echo "$node :"
    sudo ssh $node "sudo -iu postgres psql -c 'SHOW max_connections'"
done

pg1 :
max_connections
-----
123
(1 ligne)

pg2 :
max_connections
-----
123
(1 ligne)

pg3 :
max_connections
-----
123
(1 ligne)
```

L'application d'un paramètre qui ne nécessite pas de redémarrage est transparente, le rechargement de la configuration sur tous les nœuds est automatiquement déclenché par Patroni.

6/ Sauvegardes

- Configuration
- Détermination du primaire (facultatif)
 - Installation d'etcd-client
- Archivage
- Sauvegarde

6.1 CONFIGURATION SERVEUR DE PGBACKREST

La configuration se fait dans le fichier `/etc/pgbackrest.conf` :

```
# /etc/pgbackrest.conf
# S E R V E U R

[global]
repo1-path=/var/lib/pgbackrest
repo1-retention-full=2
start-fast=y
log-level-console=detail

[16-main]
pg1-path=/var/lib/postgresql/16/main
pg1-host-user=postgres
pg1-user=postgres
pg1-port=5432
```

Rendre possible l'écriture des sauvegardes et des logs :

```
root@backup:~# chown postgres: /var/log/pgbackrest
```

6.2 CONFIGURATION CLIENT DE PGBACKREST

Sur chacun des nœuds Patroni, il faut configurer le *stanza* et l'initialiser :

```
# /etc/pgbackrest.conf
# C L I E N T

[16-main]
pg1-path=/var/lib/postgresql/16/main
pg1-socket-path=/var/run/postgresql
pg1-port=5432

[global]
log-level-file=detail
log-level-console=detail
repo1-host=backup
repo1-host-user=postgres
```

Tous les nœuds doivent permettre la connexion *ssh* sans mot de passe, le *playbook Ansible* nommé `exchange_ssh_keys` permet de faire ce travail rapidement :

```
dalibo@vm:~$ sudo ansible-playbook -i inventory.yml exchange_ssh_keys.yml -f 7
```

6.2.1 Création d'une stanza

Nous pouvons alors tenter de créer la *stanza* sur le primaire :

```
postgres@pg1:~$ pgbackrest --stanza 16-main stanza-create

2024-02-01 08:12:59.812 P00    INFO: stanza-create for stanza '16-main' repo1
2024-02-01 08:13:00.547 P00    INFO: stanza-create command end: completed
↪ successfully (1369ms)

postgres@pg1:~$ pgbackrest --stanza 16-main check

ERROR: [087]: archive_mode must be enabled
2024-02-01 08:16:45.800 P00    INFO: check command end: aborted with exception [087]
```

6.2.2 Configuration de l'archivage

Toutes les instances doivent être en mesure d'archiver leurs journaux de transactions au moyen de `pgBackRest` en cas de promotion:

```
postgres@pg1:~$ patronictl edit-config
```

```
postgresql:
  parameters:
    max_connections: 123
    archive_mode: 'on'
    archive_command: pgbackrest --stanza=16-main archive-push %p
```

Notre configuration n'a pas encore été appliquée sur les instances car un redémarrage est requis :

```
postgres@pg1:~$ patronictl list -e
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+-----+-----+
| Member | Host      | Role   | State | TL | Lag in MB | Pending restart | Tags |
+-----+-----+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Leader | running | 8 |          | *              |      |
| pg2    | 10.0.3.202 | Replica | running | 8 |          0 | *              |      |
| pg3    | 10.0.3.203 | Replica | running | 8 |          0 | *              |      |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
postgres@pg1:~$ patronictl restart 16-main --force
```

```
+ Cluster: 16-main (7330283094014338096) -+-----+-----+-----+-----+
| Member | Host      | Role   | State | TL | Lag in MB | Pending restart |
+-----+-----+-----+-----+-----+-----+-----+
| pg1    | 10.0.3.201 | Leader | running | 8 |          | *              |
| pg2    | 10.0.3.202 | Replica | running | 8 |          0 | *              |
| pg3    | 10.0.3.203 | Replica | running | 8 |          0 | *              |
+-----+-----+-----+-----+-----+-----+-----+
```

```
Success: restart on member pg1
Success: restart on member pg3
Success: restart on member pg2
```

Test de la configuration de l'archivage sur le nœud pg1 :

```
postgres@pg1:~$ pgbackrest --stanza 16-main --log-level-console detail check
```

```
2024-02-01 08:26:48.829 P00 INFO: check command begin 2.50: --exec-id=6178-2f3e189a
--log-level-console=detail --log-level-file=detail
--pg1-path=/var/lib/postgresql/16/main --pg1-port=5432 --pg1-socket-path=/var/run/postgresql
--repol-host=backup --repol-host-user=postgres --stanza=16-main
2024-02-01 08:26:49.443 P00 INFO: check repo1 configuration (primary)
2024-02-01 08:26:50.294 P00 INFO: check repo1 archive for WAL (primary)
2024-02-01 08:26:51.405 P00 INFO: WAL segment 00000001000000000000000000000008 successfully archived to
'/var/lib/pgbackrest/archive/16-main/16-1/0000000100000000/00000001000000000000000000000008-06f396011ca
64c75b654ca972d55986df14f4d8b.gz' on repo1
2024-02-01 08:26:51.506 P00 INFO: check command end: completed successfully (2685ms)
```

6.3 TEST D'UNE SAUVEGARDE

Nous proposons de déclencher la sauvegarde sur le primaire, en déterminant le leader Patroni via l'interrogation de l'API etcd.

6.3.1 Détermination de l'instance primaire

Installer le paquet client etcd :

```
dalibo@vm:~$ sudo apt-get install -y etcd-client
```

Modifier le `.profile` de l'utilisateur postgres en ajoutant :

```
export ETCDCTL_ENDPOINTS='e1:2379,e2:2379,e3:2379'
```

Sur la machine backup, créer un script `leader.sh` pour déterminer qui est le *leader* Patroni, avec le contenu suivant :

```
#!/bin/bash
SCOPE='16-main'
etcdctl get /postgresql-common/${SCOPE}/leader --print-value-only
```

Donner les droits d'exécution :

```
postgres@backup:~$ chmod u+x ~/leader.sh
```

6.3.1.1 Déclenchement de la sauvegarde

Exécution d'une sauvegarde complète de l'instance (*basebackup*) :

```
postgres@backup:~$ pgbackrest --stanza 16-main --pg1-host=$(~/leader.sh) backup
↳ --type=full
```

```
2024-02-01 08:30:20.110 P00 INFO: backup command begin 2.50: --exec-id=1564-d463c2cc
--log-level-console=detail --log-level-file=off
--pg1-host=pg1 --pg1-host-user=postgres --pg1-path=/var/lib/postgresql/16/main
--pg1-port=5432 --pg1-user=postgres
--repo1-path=/var/lib/pgbackrest --repo1-retention-full=2 --stanza=16-main
--start-fast --type=full
2024-02-01 08:30:21.346 P00 INFO: execute non-exclusive backup start: backup begins after the requested
immediate checkpoint completes
2024-02-01 08:30:23.658 P00 INFO: backup start archive = 000000010000000000000000A, lsn = 0/A000060
2024-02-01 08:30:23.658 P00 INFO: check archive for prior segment 0000000100000000000000009
2024-02-01 08:30:24.743 P01 DETAIL: backup file pg1:/var/lib/postgresql/16/main/base/5/1255 (784KB, 3.46%)
checksum 83db199f681514238c9584c3d93fd800376ff957
[...]
2024-02-01 08:30:54.921 P01 DETAIL: backup file pg1:/var/lib/postgresql/16/main/base/1/13399 (0B, 100.00%)
```

```
2024-02-01 08:30:55.038 P00 INFO: execute non-exclusive backup stop and wait for all WAL segments to archive
2024-02-01 08:30:55.748 P00 INFO: backup stop archive = 000000010000000000000000A, lsn = 0/A000170
2024-02-01 08:30:56.226 P00 DETAIL: wrote 'backup_label' file returned from backup stop function
2024-02-01 08:30:56.234 P00 INFO: check archive for segment(s)
                                000000010000000000000000A:000000010000000000000000A
2024-02-01 08:30:56.500 P00 INFO: new backup label = 20240201-083021F
2024-02-01 08:30:56.838 P00 INFO: full backup size = 22.1MB, file total = 965
2024-02-01 08:30:56.838 P00 INFO: backup command end: completed successfully (36733ms)
2024-02-01 08:30:56.840 P00 INFO: expire command begin 2.50: --exec-id=1564-d463c2cc
                                --log-level-console=detail --log-level-file=off
                                --repol-path=/var/lib/pgbackrest --repol-retention-full=2
                                --stanza=16-main
2024-02-01 08:30:57.012 P00 INFO: expire command end: completed successfully (174ms)
```

Vérification de l'état de la sauvegarde :

```
postgres@backup:~$ pgbackrest --stanza 16-main info
```

```
stanza: 16-main
  status: ok
  cipher: none

db (current)
  wal archive min/max (16): 0000000100000000000000001/000000010000000000000000A

  full backup: 20240201-083021F
    timestamp start/stop: 2024-02-01 08:30:21-08 / 2024-02-01 08:30:55-08
    wal start/stop: 000000010000000000000000A / 000000010000000000000000A
    database size: 22.1MB, database backup size: 22.1MB
    repol: backup set size: 2.9MB, backup size: 2.9MB
```


7/ Références

- etcd : <https://etcd.io/docs/>
- Patroni : <https://patroni.readthedocs.io/en/latest/>
- Dalibo : <https://dalibo.com>

Notes

Notes

Notes

Nos autres publications

FORMATIONS

- DBA1 : Administration PostgreSQL
<https://dali.bo/dba1>
- DBA2 : Administration PostgreSQL avancé
<https://dali.bo/dba2>
- DBA3 : Sauvegarde et réplication avec PostgreSQL
<https://dali.bo/dba3>
- DEVPG : Développer avec PostgreSQL
<https://dali.bo/devpg>
- PERF1 : PostgreSQL Performances
<https://dali.bo/perf1>
- PERF2 : Indexation et SQL avancés
<https://dali.bo/perf2>
- MIGORPG : Migrer d'Oracle à PostgreSQL
<https://dali.bo/migorgpg>
- HAPAT : Haute disponibilité avec PostgreSQL
<https://dali.bo/hapat>

LIVRES BLANCS

- Migrer d'Oracle à PostgreSQL
<https://dali.bo/dlb01>
- Industrialiser PostgreSQL
<https://dali.bo/dlb02>
- Bonnes pratiques de modélisation avec PostgreSQL
<https://dali.bo/dlb04>
- Bonnes pratiques de développement avec PostgreSQL
<https://dali.bo/dlb05>

TÉLÉCHARGEMENT GRATUIT

Les versions électroniques de nos publications sont disponibles gratuitement sous licence open source ou sous licence Creative Commons.

8/ DALIBO, L'Expertise PostgreSQL

Depuis 2005, DALIBO met à la disposition de ses clients son savoir-faire dans le domaine des bases de données et propose des services de conseil, de formation et de support aux entreprises et aux institutionnels.

En parallèle de son activité commerciale, DALIBO contribue aux développements de la communauté PostgreSQL et participe activement à l'animation de la communauté francophone de PostgreSQL. La société est également à l'origine de nombreux outils libres de supervision, de migration, de sauvegarde et d'optimisation.

Le succès de PostgreSQL démontre que la transparence, l'ouverture et l'auto-gestion sont à la fois une source d'innovation et un gage de pérennité. DALIBO a intégré ces principes dans son ADN en optant pour le statut de SCOP : la société est contrôlée à 100 % par ses salariés, les décisions sont prises collectivement et les bénéfices sont partagés à parts égales.

