Atelier Industrialisation PostgreSQL

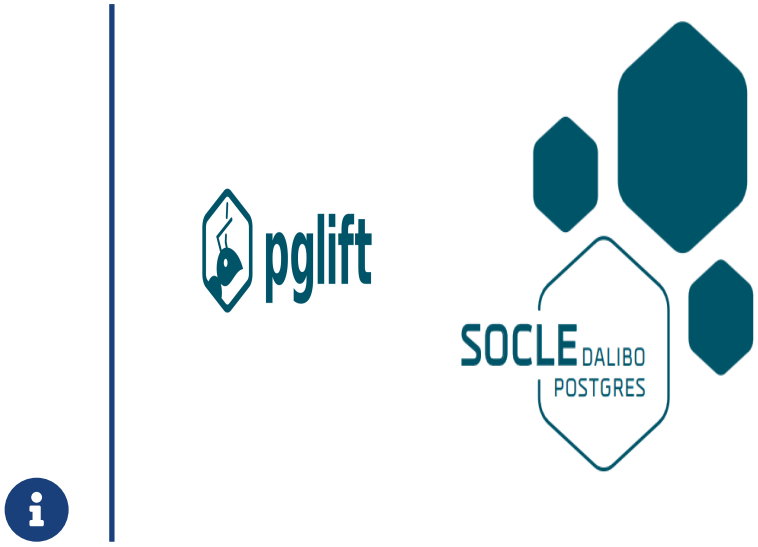
omatiser le déploiement de nœuds patroni avec

25.01

Contents

1/ Presentation

1.1 INTRODUCTION



1.2 PRÉSENTATION DE PGLIFT

- Permet de déployer et de gérer des instances *PostgreSQL* uniformisées
- Instances prêtes pour la production dès leur déploiement
- Capable de déployer des instances en réplication avec *patroni*
- Prends en charge *pgBackRest* en mode local ou distant

pglift est un outil qui permet de déployer et de gérer des instances *PostgreSQL* uniformisées. Les instances peuvent être prêtes pour la production dès leur déploiement, c'est-à-dire qu'elles sont installées avec une sauvegarde configurée et un *endpoint* de supervision accessible.

Pour les besoins de haute disponibilité, *pglift* est capable de déployer des instances en réplication avec *patroni*.

Pour effectuer des sauvegardes physiques, *pglift* prend en charge *pgBackRest* en mode local ou distant.

Par défaut, *pglift* se contente de déployer et de gérer *PostgreSQL*, les composants pris en charge sont optionnels, à activer dans sa configuration.

1.3 PRÉSENTATION DE PATRONI

- Gestionnaire de cluster *PostgreSQL*
- Assure la haute disponibilité
- Supervision et bascule automatique en cas d'incident
- Configure et maintient la réplication physique entre les instances
- Repose sur un DCS extérieur pour partager l'état des nœuds et leur configuration au sein du cluster

Patroni est un gestionnaire de cluster *PostgreSQL*, capable d'en assurer la haute disponibilité de service. Il maintient l'agrégat en condition opérationnelle, le supervise et provoque une bascule automatique en cas d'incident. Chaque démon *patroni* a la responsabilité de créer, configurer, démarrer, superviser, arrêter, promouvoir ou rétrograder son instance locale, en fonction des évènements détectés au sein du cluster. L'application des modifications de la configuration de *PostgreSQL* est effectuée par *Patroni* qui se charge de la répercuter sur tous les nœuds.

Patroni se base sur la réplication physique native de *PostgreSQL* pour assurer la haute disponibilité des données. Maîtrisant la création et configuration des instances, il est capable d'assurer automatiquement la mise en œuvre de cette réplication._Patroni_ configure et maintient cette réplication physique en mode synchrone ou asynchrone, avec ou sans cascade de réplication, en fonction de la configuration demandée. Suite à une bascule, il reconfigure automatiquement les secondaires afin que ceux-ci se reconnectent au nouveau primaire. Optionnellement, il est aussi capable de resynchroniser un ancien primaire en tant que secondaire suite à un incident.

Les démons *patroni* s'appuient sur un stockage de données distribué (DCS, Distributed Consensus Store) pour partager l'état des nœuds et leur configuration au sein du cluster. Le DCS est la « source de vérité » du cluster, le notaire arbitrant de façon fiable et officielle le leader du cluster. Les processus *patroni* lui font entièrement confiance et respectent scrupuleusement les informations qu'ils y trouvent.

1.4 PRÉSENTATION DU DCS

- Composant critique de tout cluster *Patroni*
- Choix d'*etcd* pour sa simplicité, sa robustesse et sa popularité
- Basé sur l'algorithme *Raft* (Replicated And Fault Tolerant)
- Pour stocker :
  - l'état de chaque nœud PostgreSQL
  - La configuration des instances
- Source de vérité du cluster *patroni*
- Le DCS ne doit pas devenir un *SPoF* (single point of failure)

Le DCS est un composant critique de tout cluster *Patroni*, nous avons choisi ici d'approfondir *etcd* car il est simple, robuste et populaire.

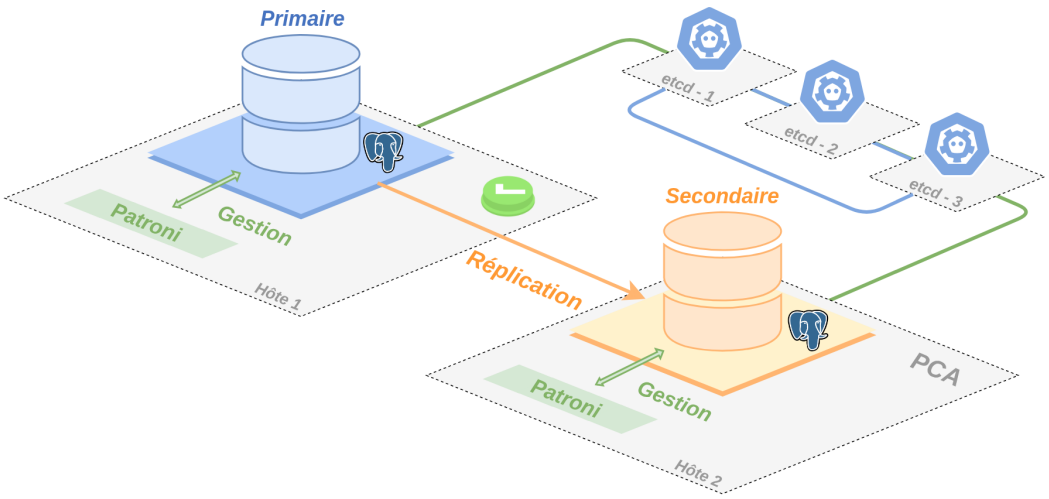
etcd est basé sur *Raft*, un algorithme de consensus répliqué et tolérant aux pannes (Replicated And Fault Tolerant). Ce genre d'algorithme vise à permettre à un ensemble de serveurs de fonctionner comme un groupe cohérent pouvant survivre à la disparition d'un certain nombre de membres.

Dans notre infrastructure, les clients *d'etcd* sont les processus *Patroni*, et leurs requêtes correspondent à : - un changement d'état du nœud ; - une modification de configuration d'une instance.

Le DCS est la « source de vérité » du cluster. Les processus *patroni* lui font entièrement confiance et respectent scrupuleusement les informations qu'ils y trouvent. À tel point que par défaut, si le leader *Patroni* ne peut plus joindre le DCS, il rétrograde immédiatement son instance locale en secondaire. Si cette opération échoue ou est trop longue, il est même capable de déclencher un reset du serveur.

Le but étant la haute disponibilité, le DCS ne doit pas devenir un *SPoF* (single point of failure). Il doit donc lui aussi être déployé en agrégat afin d'assurer une tolérance aux pannes et une disponibilité maximale du service.

1.5 ARCHITECTURE FINALE



2/ Configuration

2.1 PRÉ-REQUIS

- Un cluster *etcd*
- PostgreSQL, pgBackRest, pglift et patroni
- Activer le watchdog (recommandé)

2.2 CONFIGURATION DE PGLIFT

- Fichier de configuration *pglift* :
  - `~/.config/pglift/settings.yaml`
- Template de configuration PostgreSQL (`postgresql.conf` et `pg_hba.conf`)
- Installer la configuration de site : `pglift site-configure install`

Créer le fichier de configuration suivant dans `~/.config/pglift/settings.yaml` en adaptant les addresses IP des serveurs *etcd* et le nom des certificats.

```
---
cli:
  audit:
    path: '/pgdata/log/pglift'
postgresql:
  default_version: '17'
  datadir: '/pgdata/{version}/{name}/data'
  waldir: '/pgdata/{version}/{name}/pgwal'
  logpath: '/pgdata/log/postgresql'
  initdb:
    data_checksums: 'true'
  auth:
    local: 'peer'
    host: 'scram-sha-256'
  socket_directory: '/var/run/postgresql'
  surole:
    name: 'postgres'
  replrole: 'replication'
  dumps_directory: '/pgdata/backup/dumps/{version}-{name}'
systemd:
  user: true
  sudo: false
pgbackrest:
  configpath: /etc/pgbackrest/
  logpath: /var/log/pgbackrest/
  repository:
    mode: path
    path: '/var/lib/pgsql/backups/pitr'
    retention:
      full: 3
patroni:
  execpath: /usr/bin/patroni
```

```
  configpath: /etc/patroni/{name}.yaml
  logpath: /var/log/patroni
  etcd:
    hosts:
    - srv-pg1:2379
    - srv-pg2:2379
    - srv-pg3:2379
    protocol: https
    cacert: "/etc/pki/patroni/ca-cert.pem"
    cert: "/etc/pki/patroni/server-cert.pem"
    key: "/etc/pki/patroni/server-key.pem"
  restapi:
    cafile: "/etc/pki/patroni/ca-cert.pem"
    certfile: "/etc/pki/patroni/server-cert.pem"
    keyfile: "/etc/pki/patroni/server-key.pem"
    verify_client: required
  ctl:
    certfile: "/etc/pki/patroni/server-cert.pem"
    keyfile: "/etc/pki/patroni/server-key.pem"
  postgresql:
    use_pg_rewind: true
    passfile: /home/postgres/{name}.pgpass
```

Créer un fichier template `pg_hba.conf` dans `~/.config/pglift/postgresql/pg_hba.conf` :

```
!include include/conf/pg_hba.conf.j2
```

Installer la configuration de site `pglift` :

```
[postgres@srv-pg1 ~]$ pglift site-configure install
INFO     installed pglift-patroni@.service systemd unit at
         /home/postgres/.local/share/systemd/user/pglift-patroni@.service
INFO     installed pglift-backup@.service systemd unit at
         /home/postgres/.local/share/systemd/user/pglift-backup@.service
INFO     installed pglift-backup@.timer systemd unit at
         /home/postgres/.local/share/systemd/user/pglift-backup@.timer
INFO     installed pglift-postgresql@.service systemd unit at
         /home/postgres/.local/share/systemd/user/pglift-postgresql@.service
INFO     creating base pgBackRest configuration directory:
         /home/postgres/.local/share/pglift/etc/pgbackrest
INFO     installing base pgBackRest configuration
INFO     creating pgBackRest include directory
INFO     creating pgBackRest repository backups and archive directory:
         /pgdata/backup/pgbackrest
INFO     creating pgBackRest log directory:
         /home/postgres/.local/share/pglift/log/pgbackrest
INFO     creating pgBackRest spool directory:
```

```
         /home/postgres/.local/share/pglift/srv/pgbackrest/spool
INFO     creating PostgreSQL log directory: /pgdata/log/postgresql
```

3/ Déploiement d'instances en Haute Disponibilité avec pglift

- Via le terminal avec `pglift`
- Via un *playbook Ansible* et la collection `dalibo.pglift`

Deux méthodes de déploiement sont utilisables, via la ligne de commande ou avec un *playbook Ansible*.

Dans les exemples ci-dessous, l'utilisateur système `postgres` est utilisé.

3.1 CLI

3.1.1 Serveur primaire

Depuis `srv-pg1` :

```
[postgres@srv-pg1 ~]$ pglift instance create main --pgbackrest-stanza=main-app \
         --pgbackrest-password Passw0rd \
         --replrole-password PasswOrd \
         --surole-password PasswOrd \
         --patroni-cluster maincluster \
         --patroni-node $(hostname -s) \
         --patroni-restapi-connect-address "$(hostname -I | cut -f1 -d' '):8008" \
         --patroni-restapi-listen "$(hostname -I | cut -f1 -d' '):8008" \
         --patroni-postgresql-connect-host $(hostname -I | cut -f1 -d' ')
```

3.1.2 Serveur secondaire

Depuis `srv-pg2` :

```
[postgres@srv-pg2 ~]$ pglift instance create main --pgbackrest-stanza=main-app \
         --pgbackrest-password Passw0rd \
         --replrole-password PasswOrd \
         --surole-password PasswOrd \
         --patroni-cluster maincluster \
         --patroni-node $(hostname -s) \
         --patroni-restapi-connect-address "$(hostname -I | cut -f1 -d' '):8008" \
         --patroni-restapi-listen "$(hostname -I | cut -f1 -d' '):8008" \
         --patroni-postgresql-connect-host $(hostname -I | cut -f1 -d' ')
```

3.1.3 Lister les instances

```
[postgres@srv-pg1 ~]$ pglift instance exec main -- patronictl list
+ Cluster: maincluster (7334815726709754190) ----------+----+-----------+
| Member  | Host            | Role         | State     | TL | Lag in MB |
+---------+-----------------+--------------+-----------+----+-----------+
```

```
| srv-pg1 | 192.168.121.172 | Leader       | running   |  1 |           |
| srv-pg2 | 192.168.121.89  | Sync Standby | streaming |  1 |         0 |
+---------+-----------------+--------------+-----------+----+-----------+
```

4/ Manipuler ses nœuds patroni

- `pglift instance env`
- `pglift instance shell`

Dans un premier temps voyons la commande `pglift instance env` :

```
$ pglift instance env
PATH=/usr/pgsql-
 ↪  17/bin:/home/postgres/.local/bin:/home/postgres/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/s
 ↪  17/bin
PATRONICTL_CONFIG_FILE=/etc/patroni/17-main.yaml
PATRONI_NAME=srv-pg1
PATRONI_SCOPE=maincluster
PGBACKREST_CONFIG_PATH=/etc/pgbackrest
PGBACKREST_STANZA=main-stz
PGDATA=/pgdata/17/main/data
PGHOST=/var/run/postgresql
PGPASSFILE=/home/postgres/.pgpass
PGPORT=5432
PGUSER=postgres
PSQLRC=/pgdata/17/main/data/.psqlrc
PSQL_HISTORY=/pgdata/17/main/data/.psql_history
```

Celle-ci retourne l'ensemble des variables d'environnement qui faciliteront la gestion de nos instances PostgreSQL et nœuds Patroni.

La variable d'environnement PATRONICTL_CONFIG est alors présente permettant d'utiliser l'utilitaire `patronictl` sans avoir à préciser le fichier de configuration.

Il est alors possible de charger ces variables d'environnement facilement avec la commande `pglift instance shell`.

Une fois cette commande passée, nous pouvons utiliser l'utilitaire `patronictl`

4.1 LISTER LES NŒUDS D'UN CLUSTER

- patronictl list

La sous commande `list` de `patronictl` nous renvoie la liste des nœuds du cluster avec quelques informations intéressantes :

```
$ patronictl list
+ Cluster: maincluster (7592592231458744010)
 ↪  ---+----+-------------+-----+------------+-----+
|   Member   |    Host    |   Role  |   State   | TL | Receive LSN | Lag | Replay LSN
 ↪  | Lag |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
| srv-pg1 | srv-pg1 | Leader  | running   |  1 |             |     |           |     |
| srv-pg2 | srv-pg2 | Replica | streaming |  1 |   0/3051B58 |   0 |  0/3051B58 |   0 |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
```

La sous commande `topology` renvoie à peu près les mêmes informations, cependant la liste des nœuds change, permettant de voir quels sont les réplicas rattachés à notre instance primaire.

```
$ patronictl topology
+ Cluster: maincluster (7592592231458744010)
 ↪  -----+----+-------------+-----+------------+-----+
|    Member    |    Host    |   Role  |   State   | TL | Receive LSN | Lag | Replay
 ↪  LSN | Lag |
+--------------+------------+---------+-----------+----+-------------+-----+--------
 ↪  ----+-----+
| srv-pg1    | srv-pg1 | Leader  | running   |  1 |             |     |           |     |
| + srv-pg2 | srv-pg2 | Replica | streaming |  1 |   0/3051B58 |   0 |  0/3051B58 |
 ↪  0 |
+--------------+------------+---------+-----------+----+-------------+-----+--------
 ↪  ----+-----+
```

4.2 EFFECTUER UN SWITCHOVER

- patronictl switchover

Il est possible de programmer un *switchover* (changement de rôle de nos nœuds) à l'aide de la sous commande `switchover`. Celle-ci est interactive :

```
$ patronictl switchover
Current cluster topology
+ Cluster: maincluster (7592592231458744010)
 ↪  ---+----+-------------+-----+------------+-----+
|   Member   |    Host    |   Role   |   State   | TL | Receive LSN | Lag | Replay LSN
 ↪  | Lag |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
| srv-pg1 | srv-pg1 | Leader  | running   |  1 |             |     |           |     |
| srv-pg2 | srv-pg2 | Replica | streaming |  1 |   0/3051C60 |   0 |  0/3051C60 |   0 |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
Primary [srv-pg1]:
Candidate ['srv-pg2'] []: srv-pg2
When should the switchover take place (e.g. 2026-01-07T13:21 )  [now]:
Are you sure you want to switchover cluster maincluster, demoting current leader
 ↪  srv-pg1? [y/N]: y
2026-01-07 12:21:23.64178 Successfully switched over to "srv-pg2"
+ Cluster: maincluster (7592592231458744010)
 ↪  -+----+-------------+-----+------------+-----+
|   Member   |    Host    |   Role   |  State  | TL | Receive LSN | Lag | Replay LSN |
 ↪  Lag |
+------------+------------+---------+---------+----+-------------+-----+------------
 ↪  +-----+
| srv-pg1 | srv-pg1 | Replica | stopped |    |     unknown |     |    unknown |     |
| srv-pg2 | srv-pg2 | Leader  | running |  1 |             |     |            |     |
+------------+------------+---------+---------+----+-------------+-----+------------
 ↪  +-----+
```

Notre instance primaire est alors reconstruite en instance secondaire et l'instance secondaire est promue Leader (instance primaire).

Cette opération peut être utile lors de mises à jour mineures de PostgreSQL, mettant à jour d'abord les instances secondaires, suivi d'un *switchover* avant de mettre à jour l'instance primaire.

4.3 TESTER LE FAILOVER (BASCULE AUTOMATIQUE)

- `pkill -9 patroni`
- Redémarrage du noeud si le Watchdog est activé

En cas de problème sur notre instance primaire, *patroni* se charge d'effectuer automatiquement la promotion d'une instance secondaire en instance primaire.

Si le `watchdog` est activé, le serveur avec l'instance primaire sera alors redémarré et *patroni* se chargera de la reconstruction de l'instance pour rejoindre de nouveau le cluster en tant qu'instance secondaire.

Pour simuler une panne, il suffit de stopper *patroni* sur le nœud principal :

```
$ pkill -9 patroni
```

Sur un des nœuds secondaire, la commande `patronictl list` nous montre bien le changement de nœud Leader :

```
$ patronictl list
+ Cluster: maincluster (7592592231458744010)
 ↪  +----+-------------+-----+------------+-----+
|   Member   |    Host    |  Role  |  State  | TL | Receive LSN | Lag | Replay LSN |
 ↪  Lag |
+------------+------------+--------+---------+----+-------------+-----+------------
 ↪  +-----+
| srv-pg1 | srv-pg1 | Leader | running |  3 |             |     |            |     |
+------------+------------+--------+---------+----+-------------+-----+------------
 ↪  +-----+
```

4.4 ÉDITER LA CONFIGURATION POSTGRESQL

```
- patronictl show-config
- pglift pgconf show max_connections
- patronictl edit-config
```

Certains paramètres PostgreSQL sont gérés exclusivement par *patroni* comme par exemple `max_connections`. Afin de changer la valeur de ce paramètre, *patroni* permet de le faire avec la sous commande `edit-config`. Il est également possible de changer des paramètres *patroni* à l'aide de cette sous commande.

Avant de modifier la valeur de `max_connections`, voyons les paramètres déjà en place :

```
$ patronictl show-config
loop_wait: 10
```

Avec pglift, vérifions la valeur de `max_connections`:

```
$ pglift pgconf show max_connections
max_connections = '100'
```

Changeons maintenant la valeur de `max_connections` avec `patronictl`

```
$ patronictl edit-config
loop_wait: 10
postgresql:
    parameters:
        max_connections: 200
```

La commande `patronictl list` nous montre alors que le paramètre est changé mais nécessite un redémarrage.

```
+ Cluster: maincluster (7592592231458744010) ---+----+-------------+-----+----------
 ↪  --+-----+-----------------+---------------------------+
|   Member   |    Host    |   Role  |   State   | TL | Receive LSN | Lag | Replay LSN
 ↪  | Lag | Pending restart |   Pending restart reason  |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+-----------------+---------------------------+
| srv-pg1 | srv-pg1 | Leader  | running   |  3 |             |     |           |     |
 ↪  *               | max_connections: 100->200 |
| srv-pg2 | srv-pg2 | Replica | streaming |  3 |   0/4000320 |   0 |  0/4000320 |   0
 ↪  | *               | max_connections: 100->200 |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+-----------------+---------------------------+
```

4.5 REDÉMARRER LES NŒUDS DU CLUSTER

- patronictl restart CLUSTER-NAME

La sous commande `restart` permet de redémarrer l'ensemble des services patroni des nœuds d'un cluster.

Afin d'appliquer le changement de configuration fait précédemment, redémarrons avec cette sous commande :

```
$ patronictl restart maincluster
+ Cluster: maincluster (7592592231458744010) ---+----+------------+-----+----------
 ↪  --+-----+-----------------+--------------------------+
|   Member   |    Host    |   Role  |   State   | TL | Receive LSN | Lag | Replay LSN
 ↪  | Lag | Pending restart |   Pending restart reason   |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+-----------------+--------------------------+
| srv-pg1 | srv-pg1 | Leader  | running   |  3 |             |     |            |     |
 ↪  *               | max_connections: 100->200 |
| srv-pg2 | srv-pg2 | Replica | streaming |  3 |   0/4000320 |   0 |  0/4000320 |   0
 ↪  | *               | max_connections: 100->200 |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+-----------------+--------------------------+
When should the restart take place (e.g. 2026-01-07T13:28)  [now]: now
Are you sure you want to restart members srv-pg1, srv-pg2? [y/N]: y
Restart if the PostgreSQL version is less than provided (e.g. 9.5.2)  []:
Success: restart on member srv-pg1
Success: restart on member srv-pg2
```

En relançant la commande `patronictl list`, on peut voir que tout les nœuds n'ont plus le statut "Pending Restart".

```
$ patronictl list
+ Cluster: maincluster (7592592231458744010)
 ↪  ---+----+-------------+-----+------------+-----+
|   Member   |    Host    |   Role  |   State   | TL | Receive LSN | Lag | Replay LSN
 ↪  | Lag |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
| srv-pg1 | srv-pg1 | Leader  | running   |  3 |             |     |            |     |
| srv-pg2 | srv-pg2 | Replica | streaming |  3 |   0/5000110 |   0 |  0/5000110 |   0 |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
```

4.6 SUPPRIMER LES NŒUDS D'UN CLUSTER

- `pglift instance drop`

`pglift` permet la suppression d'instance PostgreSQL avec la sous commande `pglift instance drop`. La suppression de l'instance entraîne automatiquement la suppression du nœud dans le cluster patroni.

Il suffit alors de lancer cette commande sur l'ensemble des nœuds. A noter, lors de la suppression du dernier nœud du cluster, `pglift` conserve une copie du fichier de configuration de patroni. Cela permettra par la suite de supprimer le cluster défini dans notre DCS `etcd`.

```
$ pglift instance drop
INFO     dropping instance 17/main
> Confirm complete deletion of instance 17/main? [y/n] (y): y
INFO     stopping systemd unit pglift-backup@17-main.timer
INFO     deconfiguring Prometheus postgres_exporter 17-main
INFO     removing entries matching port=5432 from /home/postgres/.pgpass
WARNING  'srv-pg1' appears to be the last member of cluster 'maincluster',
         saving Patroni configuration file to
         /etc/patroni/maincluster-srv-pg1-1767789122.79803.yaml; see
         https://pglift.readthedocs.io/en/latest/user/ops/ha.html#cluster-remova
         l for more information
INFO     stopping Patroni 17-main
INFO     stopping systemd unit pglift-patroni@17-main.service
INFO     deconfiguring Patroni service
INFO     deleting PostgreSQL data and WAL directories
```

4.7 SUPPRESSION DU CLUSTER

- `patronictl -c CONFIG-FILE remove CLUSTER-NAME`

La suppression du cluster ne peut se faire uniquement si aucun nœud n'est présent (et donc supprimé comme vu plus haut). Cette action est faite en appelant la sous commande `remove` de `patronictl`. Cependant comme le nœud a été supprimé, il est nécessaire pointer le fichier de configuration sauvegardé par pglift :

```
$ patronictl -c /etc/patroni/maincluster-srv-pg1-1767789122.79803.yaml remove
 ↪  maincluster
+ Cluster: maincluster (7592596391215995957) -----+-----+------------+-----+
| Member | Host | Role | State | TL | Receive LSN | Lag | Replay LSN | Lag |
+--------+------+------+-------+----+-------------+-----+------------+-----+
+--------+------+------+-------+----+-------------+-----+------------+-----+
Please confirm the cluster name to remove: maincluster
You are about to remove all information in DCS for maincluster, please type: "Yes I
 ↪  am aware": Yes I am aware
```

Si on liste de nouveau, on peut voir le statut du cluster en `uninitialized`.

```
$ patronictl -c /etc/patroni/maincluster-srv-pg1-1767789122.79803.yaml list
+ Cluster: maincluster (uninitialized) -----------+-----+------------+-----+
| Member | Host | Role | State | TL | Receive LSN | Lag | Replay LSN | Lag |
+--------+------+------+-------+----+-------------+-----+------------+-----+
+--------+------+------+-------+----+-------------+-----+------------+-----+
```

5/ Utiliser la configuration dynamique de patroni

- Par défaut, configuration locale
- Configuration dynamique (stocké dans le DCS):

```
patroni:
  configuration_mode:
    auth: dynamic
    parameters: dynamic
```

Par défaut, la configuration de PostgreSQL se fait localement, sauf pour quelques paramètres nécessitant être présent sur le DCS. Cela est géré par `pglift` avec cette configuration :

```
patroni:
  configuration_mode:
    auth: local
    parameters: local
```

Il est alors possible d'utiliser la sous commande `pglift pgconf edit` pour changer la configuration localement et `pglift pghba edit` pour l'authentification. Il faudra alors appliquer la même configuration manuellement sur l'ensemble des nœuds du cluster.

Afin d'appliquer une configuration sur l'ensemble des nœuds, le mode dynamique peut être utilisé. Pour cela, il faudra appliquer cette configuration dans la section `patroni` du fichier de configuration de `pglift` (`/etc/pglift/settings.yaml`):

```
patroni:
  configuration_mode:
    auth: dynamic
    parameters: dynamic
```

5.1 DÉPLOYER UN CLUSTER PATRONI

- Adapter le fichier d'hôtes (`~/ansible/hosts`)
- `ansible-playbook -i hosts playbooks/instances.yaml`

Afin d'appliquer une configuration avec le mode dynamique, créons un cluster à l'aide de ce playbook *Ansible* :

```
---
- name: Create Instances
  hosts: database
  become_user: postgres
  become: true
  tasks:
    - name: Managing instances
      dalibo.pglift.instance:
        name: main
        state: started
        version: 17
        port: 5432
        surole_password: "Passw0rd"
        replrole_password: "Passw0rd"
        pgbackrest:
          password: "Passw0rd"
          stanza: main-stz
        patroni:
          cluster: maincluster
          node: "{{ ansible_hostname }}"
          restapi:
            connect_address: "{{ ansible_eth0.ipv4.address }}:8008"
            listen: "{{ ansible_eth0.ipv4.address }}:8008"
          postgresql:
            connect_host: "{{ ansible_default_ipv4.address  }}"
      throttle: 1
```

Une fois les nœuds créés et démarrés, la commande `patronictl show-config` nous montre bien plus de paramètres :

```
loop_wait: 10
postgresql:
  parameters:
    archive_command: /usr/bin/pgbackrest --config-path=/etc/pgbackrest
 ↪  --stanza=main-stz --pg1-path=/pgdata/17/main/data archive-push %p
    archive_mode: true
```

```
    effective_cache_size: 1 GB
    lc_messages: C
    lc_monetary: C
    lc_numeric: C
    lc_time: C
    log_destination: stderr
    log_directory: /pgdata/log/postgresql
    log_filename: 17-main-%Y-%m-%d.log
    logging_collector: true
    shared_buffers: 464 MB
    unix_socket_directories: /var/run/postgresql
    wal_level: replica
  pg_hba:
  - '# #'
  - '# Ansible managed'
  - '#'
  - local all postgres peer
  - local all backup scram-sha-256
  - local all prometheus scram-sha-256
  - local all all scram-sha-256
  - host  all all ::1/128 scram-sha-256
  - local replication replication scram-sha-256
  - host replication replication 127.0.0.1/32 scram-sha-256
  - host replication replication ::1/128 scram-sha-256
  - host replication replication 0.0.0.0/0 scram-sha-256
  - host postgres postgres 0.0.0.0/0 scram-sha-256
```

`pglift` a donc positionné les paramètres PostgreSQL au sein du DCS à l'aide de *patroni*.

En utilisant `pglift pghba edit`, pglift positionne également la configuration à l'aide de patroni :

```
$ pglift pghba edit
[...]
host    db1             bob             0.0.0.0/0               scram-sha-256

$ patronictl show-config
loop_wait: 10
postgresql:
  parameters:
    archive_command: /usr/bin/pgbackrest --config-path=/etc/pgbackrest
     ↪  --stanza=main-stz --pg1-path=/pgdata/17/main/data archive-push %p
    archive_mode: true
    effective_cache_size: 1 GB
    lc_messages: C
    lc_monetary: C
    lc_numeric: C
    lc_time: C
```

```
    log_destination: stderr
    log_directory: /pgdata/log/postgresql
    log_filename: 17-main-%Y-%m-%d.log
    logging_collector: true
    shared_buffers: 464 MB
    unix_socket_directories: /var/run/postgresql
    wal_level: replica
  pg_hba:
  - '# #'
  - '# Ansible managed'
  - '#'
  - local   all             postgres                                peer
  - local   all             backup                                  scram-sha-256
  - local   all             prometheus                              scram-sha-256
  - local   all             all                                     scram-sha-256
  - host    all             all             ::1/128                 scram-sha-256
  - local   replication     replication                             scram-sha-256
  - host    replication     replication     127.0.0.1/32            scram-sha-256
  - host    replication     replication     ::1/128                 scram-sha-256
  - host    replication     replication     0.0.0.0/0               scram-sha-256
  - host    postgres        postgres        0.0.0.0/0               scram-sha-256
  - host    db1             bob             0.0.0.0/0               scram-sha-256
```

6/ Déployer une adresse IP Virtuelle

- **keepalived**: Solution d'adresse IP Virtuelle
- `ansible-playbook -i hosts playbooks/keepalived.yaml`

L'utilisation d'une adresse IP virtuelle permet de faire en sorte que le service PostgreSQL soit accessible, quelque soit le noeud primaire. La solution la plus simple à mettre en place est celle de `keepalived`, à déployer sur chaque noeud d'un cluster *patroni*.

La collection Ansible *Extras* de Dalibo[1] contient un rôle *keepalived* permettant un déploiement simple de cette solution.

Voici un exemple d'utilisation du rôle `dalibo.extras.keepalived` :

```
---
- name: Install & configure keeplived
  hosts: database
  gather_facts: true
  become: true
  vars:
    keepalived_hosts_group: database
    keepalived_configuration_type: patroni
    keepalived_vip: 10.10.0.100
    keepalived_interface: '{{ ansible_default_ipv4.interface }}'
    keepalived_auth_password: PassW0rd
    keepalived_unicast: false
    keepalived_patroni_tls:
      cacert: "/etc/pki/tls/certs/ca_cert.pem"
      cert: "/etc/pki/tls/certs/{{ inventory_hostname }}.pem"
      key: "/etc/pki/tls/private/{{ inventory_hostname }}.key"
  roles:
    - dalibo.extras.keepalived
```

Pour savoir qui est le *Leader*, *keepalived* fait un test sur l'endpoint *patroni* appelé `primary` :

```
vrrp_script check_patroni_primary {
 script "/usr/bin/curl -X GET -I --fail --cert /etc/pki/tls/certs/pglift-
pg1.pem --key /etc/pki/tls/private/pglift-pg1.key --cacert /etc/pki/tls/certs/ca_c
  interval 2
}
```

[1] https://galaxy.ansible.com/ui/repo/published/dalibo/extras/

On peut donc voir cette adresse IP virtuelle **10.10.0.100** présente sur notre noeud *Leader* :

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 ↪  qlen 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 ↪  default qlen 1000
    link/ether 52:54:00:2e:ee:b0 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    altname ens3
    inet 192.168.122.88/24 brd 192.168.122.255 scope global dynamic noprefixroute
     ↪  eth0
       valid_lft 2929sec preferred_lft 2929sec
    inet 10.10.0.100/24 scope global eth0:1
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe2e:eeb0/64 scope link
       valid_lft forever preferred_lft forever
```

Après un switchover, l'adresse IP virtuelle est présente sur le nouveau Leader :

```
$ patronictl switchover
Current cluster topology
+ Cluster: maincluster (7592610216175489546)
 ↪  ---+----+-------------+-----+------------+-----+
|   Member   |    Host    |   Role  |   State   | TL | Receive LSN | Lag | Replay LSN
 ↪  | Lag |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
| pglift-pg1 | pglift-pg1 | Leader  | running   |  1 |             |     |
 ↪  |     |
| pglift-pg2 | pglift-pg2 | Replica | streaming |  1 |   0/3051C60 |   0 |  0/3051C60
 ↪  |   0 |
+------------+------------+---------+-----------+----+-------------+-----+----------
 ↪  --+-----+
Primary [pglift-pg1]:
Candidate ['pglift-pg2'] []: pglift-pg2
When should the switchover take place (e.g. 2026-01-07T15:13 )  [now]:
Are you sure you want to switchover cluster maincluster, demoting current leader
 ↪  pglift-pg1? [y/N]: y
2026-01-07 14:13:09.13858 Successfully switched over to "pglift-pg2"
+ Cluster: maincluster (7592610216175489546)
 ↪  -+----+-------------+-----+------------+-----+
```

```
|   Member   |    Host    |   Role   |  State   | TL | Receive LSN | Lag | Replay LSN |
 ↪  Lag |
+------------+------------+---------+---------+----+-------------+-----+------------
 ↪  +-----+
| pglift-pg1 | pglift-pg1 | Replica | stopped |    |     unknown |     |    unknown |
 ↪  |
| pglift-pg2 | pglift-pg2 | Leader  | running |  1 |             |     |            |
 ↪  |
+------------+----

$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 ↪  qlen 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 ↪  default qlen 1000
    link/ether 52:54:00:c2:c8:b2 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    altname ens3
    inet 192.168.122.180/24 brd 192.168.122.255 scope global dynamic noprefixroute
     ↪  eth0
       valid_lft 2878sec preferred_lft 2878sec
    inet 10.10.0.100/24 scope global eth0:1
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fec2:c8b2/64 scope link
       valid_lft forever preferred_lft forever
```

Et absente sur l'ancien *Leader* :

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
 ↪  qlen 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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 ↪  default qlen 1000
    link/ether 52:54:00:2e:ee:b0 brd ff:ff:ff:ff:ff:ff
    altname enp0s3
    altname ens3
    inet 192.168.122.88/24 brd 192.168.122.255 scope global dynamic noprefixroute
     ↪  eth0
```

```
       valid_lft 2772sec preferred_lft 2772sec
    inet6 fe80::5054:ff:fe2e:eeb0/64 scope link
       valid_lft forever preferred_lft forever
```

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
- 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.

7/ 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.