

Atelier Industrialisation PostgreSQL

Industrialisation avec pglift et Ansible



25.01

Contents

1/ Présentation	1
1.1 Introduction	2
1.2 Présentation de pglift	3
1.3 CLI	4
1.4 Ansible (Collection dalibo.pglift)	7
1.5 Ansible (Collections dalibo.essential, dalibo.advanced, dalibo.extras)	8
2/ Installation et création d'instance avec pglift (CLI).	9
2.1 Pré-requis	10
2.2 Installation de PostgreSQL	12
2.3 Installation de pgBackrest	13
2.4 Installation de pglift	14
2.4.1 pipx	14
2.5 Configuration initiale	15
2.6 Déploiement d'une instance (CLI)	18
3/ Installation et création d'instance avec pglift (Ansible).	21
3.1 Inventaire	22
3.2 Collection	23
3.3 Génération des certificats TLS	24
3.4 Installation de temBoard UI	25
3.5 Installation du cluster ETCD	26
3.6 Installation de l'environnement	27
3.7 Déploiement d'une instance avec Ansible	28
3.8 Création de l'instance	29
3.8.1 Création d'un secondaire en streaming réplication	30
4/ Sauvegarde et Restauration pgbackrest avec pglift	31
4.1 Rappel Point In Time Recovery (PITR)	32
4.2 Sauvegarde	33
4.3 Restauration	35
5/ Manipulation d'instances avec pglift	37
5.1 Lister les instances	38
5.2 Obtenir la descriptions des instances	39
5.3 Obtenir le statut d'une instance	40

5.4	Gestion de l'état	41
5.5	Consulter les traces	42
5.6	Gestion de la configuration de PostgreSQL	43
5.6.1	Modifier la configuration	44
5.6.2	Afficher la configuration	44
5.6.3	Supprimer une configuration	45
5.7	Gestion du pg_hba.conf	46
5.8	Maintenance des données	47
5.9	Opérations sur les roles	48
5.10	Opérations sur les bases de données	50
5.11	Réplication	52
6/	Déployer des instances en Haute Disponibilité avec pglift	55
6.1	Configuration de pglift	56
6.2	Déploiement d'instances en Haute Disponibilité avec pglift	58
6.3	CLI	59
6.4	Ansible	60
6.5	Création des noeuds du cluster	61
6.6	Gestion de la configuration du cluster	62
6.6.1	Avec Patroni	62
6.6.2	Avec pglift	62
6.7	Statut du cluster	63
6.8	Switchover	64
6.9	Failover	65
7/	Gestion de parc d'instances avec temBoard.	67
7.1	Présentation de temBoard	68
7.2	Enregistrement de l'instance	69
7.3	Visualisation des métriques	70
Notes		73
Notes		75
Notes		77
Nos autres publications		79
Formations		80
Livres blancs		81
Téléchargement gratuit		82

1/ Présentation

1.1 INTRODUCTION



pglift

SOCLE

DALIBO
POSTGRES

Cet atelier a pour but de détailler l'industrialisation d'instances PostgreSQL avec *pglift*.

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 CLI



- L'ensemble de ces fonctionnalités sont exposées dans une interface en ligne de commande

pglift expose l'ensemble de ses fonctionnalités dans une interface en ligne de commande.

```
$ pglift
```

```
Usage: pglift [OPTIONS] COMMAND [ARGS]...
```

```
Deploy production-ready instances of PostgreSQL
```

Options:

```
-L, --log-level [DEBUG|INFO|WARNING|ERROR|CRITICAL]
                                Set log threshold (default to INFO when
                                logging to stderr or WARNING when logging to
                                a file).
-l, --log-file LOGFILE          Write logs to LOGFILE, instead of stderr.
--interactive / --non-interactive
                                Interactively prompt for confirmation when
                                needed (the default), or automatically pick
                                the default option for all choices.
--version                       Show program version.
--completion [bash|fish|zsh]    Output completion for specified shell and
                                exit.
--help                           Show this message and exit.
```

Commands:

```
instance      Manage instances.
pgconf         Manage configuration of a PostgreSQL instance.
role           Manage roles.
database       Manage databases.
patroni        Handle Patroni service for an instance.
postgres_exporter Handle Prometheus postgres_exporter
```

La commande permet de créer et de gérer des instances PostgreSQL et les services associés. Elle permet également de créer des bases de données et des rôles dans une instance existante.

L'aide de chaque commande citées ci-dessus peut être affichée. Par exemple, pour l'aide de la commande `instance` :

```
$ pglift instance --help
```

```
Usage: pglift instance [OPTIONS] COMMAND [ARGS]...
```

Manage instances.

Options:

```
--schema  Print the JSON schema of instance model and exit.
--help    Show this message and exit.
```

Commands:

```
alter      Alter PostgreSQL INSTANCE
backups    List available backups for INSTANCE
create     Initialize a PostgreSQL instance
drop       Drop PostgreSQL INSTANCE
env        Output environment variables suitable to handle to...
exec       Execute command in the libpq environment for PostgreSQL...
get        Get the description of PostgreSQL INSTANCE.
list       List the available instances
logs       Output PostgreSQL logs of INSTANCE.
privileges List privileges on INSTANCE's databases.
promote    Promote standby PostgreSQL INSTANCE
reload     Reload PostgreSQL INSTANCE
restart    Restart PostgreSQL INSTANCE
restore    Restore PostgreSQL INSTANCE
shell      Start a shell with instance environment.
start      Start PostgreSQL INSTANCE
status     Check the status of instance and all satellite components.
stop       Stop PostgreSQL INSTANCE
upgrade    Upgrade INSTANCE using pg_upgrade
```

Il en va de même pour les sous-commandes, par exemple, pour l'aide de `pglift instance alter` :

```
$ pglift instance alter --help
```

```
Usage: pglift instance alter [OPTIONS] [INSTANCE]
```

Alter PostgreSQL INSTANCE

INSTANCE identifies target instance as <version>/<name> where the <version>/prefix may be omitted if there is only one instance matching <name>. Required if there is more than one instance on system.

Options:

```
--port PORT      TCP port the postgresql instance will be
                  listening to. If unspecified, default to
                  5432 unless a 'port' setting is found in
```

```
                                'settings'.
--data-checksums / --no-data-checksums    Enable or disable data checksums. If
                                           unspecified, fall back to site settings
                                           choice.
--state [started|stopped]                Runtime state.
--powa-password TEXT                     Password of PostgreSQL role for PoWA.
--prometheus-port PORT                   TCP port for the web interface and telemetry
                                           of Prometheus.
--prometheus-password TEXT               Password of PostgreSQL role for Prometheus
                                           postgres_exporter.
--patroni-restapi-connect-address CONNECT_ADDRESS
                                           IP address (or hostname) and port, to access
                                           the Patroni's REST API.
--patroni-restapi-listen LISTEN          IP address (or hostname) and port that
                                           Patroni will listen to for the REST API.
                                           Defaults to connect_address if not provided.
--patroni-postgresql-connect-host CONNECT_HOST
                                           Host or IP address through which PostgreSQL
                                           is externally accessible.
--patroni-postgresql-replication-ssl-cert CERT
                                           Client certificate.
--patroni-postgresql-replication-ssl-key KEY
                                           Private key.
--patroni-postgresql-replication-ssl-password TEXT
                                           Password for the private key.
--patroni-postgresql-rewind-ssl-cert CERT
                                           Client certificate.
--patroni-postgresql-rewind-ssl-key KEY
                                           Private key.
--patroni-postgresql-rewind-ssl-password TEXT
                                           Password for the private key.
--patroni-etcd-username USERNAME        Username for basic authentication to etcd.
--patroni-etcd-password TEXT            Password for basic authentication to etcd.
--help                                  Show this message and exit.
```

1.4 ANSIBLE (COLLECTION DALIBO.PGLIFT)



- Fonctionnalités de *pglift* accessibles depuis la collection `dalibo.pglift`
- Permet d'intégrer *pglift* dans un processus de déploiement automatisé *ansible*

Les fonctionnalités de *pglift* sont également accessibles depuis la collection `dalibo.pglift`. Celle-ci fournit les modules *ansible* permettant d'intégrer les opérations de *pglift* dans un processus de déploiement automatisé déclaratif à l'aide d'*ansible* (*infrastructure as code*).

1.5 ANSIBLE (COLLECTIONS DALIBO.ESSENTIAL, DALIBO.ADVANCED, DALIBO.EXTRAS)



- Permet une installation et un déploiement automatisé de l'ensemble des éléments nécessaires à *pglift*
- Chaque collection est spécifique à un thème :
 - `dalibo.essential` : PostgreSQL, réplication, monitoring, sauvergarde
 - `dalibo.advanced` : HA, performances, audit, ldap, pooler
 - `dalibo.extras` : Tous les outils annexes (*ETCD*, *rsyslog*, *logrotate*, etc)

Des collections Ansible publiques sont également développées par Dalibo. Elles permettent une installation et un déploiement automatisés de l'ensemble des composants nécessaires au fonctionnement de *pglift*.

Ces collections sont un regroupement de fonctionnalités par thème :

- `dalibo.essential` : Pour PostgreSQL, la réplication, le monitoring et la sauvergarde
- `dalibo.advanced` : pour la HA, les performances, les audits, ldap, et les pooler de connexions
- `dalibo.extras` : Regroupe tous les outils annexes comme *ETCD*, *rsyslog*, *logrotate*, etc.

2/ Installation et création d'instance avec pglift (CLI).

2.1 PRÉ-REQUIS



- Dépôts Powertools, EPEL, PGDG et Dalibo Labs
- Utilisateur système postgres
- Activer le **lingering**

Les machines suivantes sont utilisées pour cet atelier :

Serveur	OS	Rôle
srv-pg1	RockyLinux 8	Serveur de bases de données
srv-helper	RockyLinux 8	Serveur de sauvegarde et supervision

L'ensemble des tâches seront effectuées sur serveur `srv-pg1`.

Activer le dépôt additionnel `PowerTools` :

```
[root@srv-pg1 ~]# dnf config-manager -y --set-enabled powertools
```

Installer le dépôt *EPEL* :

```
[root@srv-pg1 ~]# dnf install -y epel-release
```

Installer le dépôt *PGDG* de la communauté PostgreSQL :

```
[root@srv-pg1 ~]# dnf install -y \
https://download.postgresql.org/pub/repos/yum/repos/EL-8-x86_64/\
pgdg-redhat-repo-latest.noarch.rpm
```

Désactiver le module `dnf postgresql` afin de pouvoir installer les paquets PostgreSQL de la communauté :

```
[root@srv-pg1 ~]# dnf -y module disable postgresql
```

Installer le dépôt *Dalibo Labs* :

```
[root@srv-pg1 ~]# dnf -y install \
https://yum.dalibo.org/labs/dalibo-labs-4-1.noarch.rpm
```

Installer *python 3.9* sur les serveurs PostgreSQL :


```
[root@srv-pg1 ~]# dnf -y install python39
```

Créer l'utilisateur postgres

```
[root@srv-pg1 ~]# useradd -U -d /home/postgres -s /bin/bash postgres
```



L'utilisateur est créé avant d'installer PostgreSQL pour avoir le contrôle sur le répertoire *home* utilisé.

Créer le répertoire pour l'arborescence des données :

```
[root@srv-pg1 ~]# mkdir /pgdata && chown postgres:postgres /pgdata
```

Créer les répertoires de configuration de *pglift* dans *~/.config* en tant que postgres :

```
[root@srv-pg1 ~]# su - postgres << EOF
mkdir -p ~/.config/pglift/postgresql
mkdir -p ~/.config/pglift/pgbackrest
EOF
```

Configurer le `linger` sur l'utilisateur postgres, ce qui lui permettra de faire fonctionner des services système, au même titre que root :

```
[root@srv-pg1 ~]# loginctl enable-linger postgres
```

2.2 INSTALLATION DE POSTGRESQL

Installer PostgreSQL 16 :

```
[root@srv-pg1 ~]# dnf install -y postgresql16 postgresql16-server \
postgresql16-contrib
```

2.3 INSTALLATION DE PGBACKREST

Installer pgBackRest :

```
[root@srv-pg1 ~]# dnf install -y pgbackrest
```

2.4 INSTALLATION DE PGLIFT

2.4.1 pipx

Installer *pglift* avec *pipx*, en tant que *postgres* :

```
[root@srv-pg1 ~]# su - postgres << EOF
pip3.9 install pipx
~/.local/bin/pipx install "pglift[cli]" --include-deps
~/.local/bin/pipx ensurepath
EOF
```

Ouvrir une nouvelle session est nécessaire pour que le binaire *pglift* soit dans le `${PATH}` de l'utilisateur *postgres*

2.5 CONFIGURATION INITIALE



- Fichier de configuration *pglift* :
 - `~/.config/pglift/settings.yaml`
 - `/etc/pglift/settings.yaml`
- Template de configuration PostgreSQL et pgBackRest
- Installer la configuration de site : `pglift site-configure install`

Le fichier de configuration principal de *pglift* déclare le fonctionnement de ses opérations. C'est un fichier au format *YAML*.

Une première version assez basique de cette configuration est à déposer dans le fichier `~/.config/pglift/settings.yaml`

```
---
systemd: {}
postgresql:
  default_version: '16'
  auth:
    local: 'peer'
    host: 'scram-sha-256'
  surole:
    name: 'postgres'
  backuprole:
    name: 'backup'
  datadir: '/pgdata/{version}/{name}/data'
  logpath: '/pgdata/log/postgresql'
  dumps_directory: '/pgdata/backup/dumps/{version}-{name}'
  socket_directory: '/var/run/postgresql/'
  replrole: replication
pgbackrest:
  repository:
    path: '/pgdata/backup/pgbackrest'
    mode: 'path'
```

pglift substitue des variables à partir des caractéristiques de l'instance à déployer. Ainsi, les variables `{version}` et `{name}`, qui sont obligatoires pour l'option `datadir`, seront remplacées par la version PostgreSQL de l'instance et le nom qui est renseigné à sa création.

En plus de ce fichier, *pglift* supporte l'utilisation de *templates* de fichiers de configuration. Ces

derniers peuvent être utilisés pour modifier globalement les paramètres associés aux instances qui seront ensuite déployées sur le nœud local.

Les templates suivants sont à déposer sur les nœuds PostgreSQL :

- `~/config/pglift/postgresql/postgresql.conf` : Configuration de l'instance PostgreSQL

```
listen_addresses = '*'
port = 5432
shared_buffers = 25%
effective_cache_size = 66%
random_page_cost = 1.5
wal_buffers = '8MB'
checkpoint_completion_target = 0.9
timezone = 'Europe/Paris'
cluster_name = {name}
unix_socket_directories = {settings.socket_directory}
logging_collector = true
log_directory = {settings.logpath}
shared_preload_libraries = 'pg_stat_statements'
```

- `~/config/pglift/postgresql/pg_hba.conf` : Configuration de l'authentification PostgreSQL

local	all	{surole}		{auth.local}
local	all	{backuprole}		{auth.local} map=backupmap
local	temboard	temboard		{auth.local}
local	all	temboardagent		{auth.local} map=temboardmap
local	all	bob		scram-sha-256
host	replication	{replrole}	127.0.0.1/32	{auth.host}
host	all	all	0.0.0.0/0	{auth.host}

- `~/config/pglift/postgresql/pg_ident.conf` : Mapping des utilisateurs système avec des rôles PostgreSQL

```
# MAPNAME      SYSTEM-USERNAME PG-USERNAME
backupmap      {sysuser}      {backuprole}
temboardmap    {sysuser}      temboardagent
```

- `~/config/pglift/pgbackrest/pgbackrest.conf` : Configuration globale de *pgBackRest*

```
[global]
lock-path = {lockpath}
log-path = {logpath}
log-level-console = info
```

pglift substitue certaines variables dans les *templates* à partir de son paramétrage, ou bien des spécifications du système. Par exemple :

- `shared_buffers = 25%` : la valeur du paramètre sera transformée en 25% de la quantité totale de mémoire sur le serveur lors du déploiement.
- `{surole}` sera remplacé par le nom du super-utilisateur choisi (postgres par défaut)
- `{auth.host}` correspondra à la valeur `postgresql.auth.host` de la configuration de *pglift*.

Avant de pouvoir créer des instances, il faut préparer le système à accueillir des instances *pglift* selon la configuration actuelle. Pour ce faire, exécuter la commande `pglift site-configure install`

```
$ pglift site-configure install
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
```

2.6 DÉPLOIEMENT D'UNE INSTANCE (CLI)



```
$ pglift instance create main \
--pgbackrest-stanza=main
```

Déployer une instance à l'aide de la ligne de commande `pglift` :

```
$ pglift instance create main --pgbackrest-stanza=main
INFO      initializing PostgreSQL
INFO      configuring PostgreSQL authentication
INFO      configuring PostgreSQL
INFO      starting PostgreSQL 16/main
INFO      creating role 'backup'
INFO      configuring pgBackRest stanza 'main' for pg1-path=/pgdata/16/main/data
INFO      creating pgBackRest stanza main
INFO      checking pgBackRest configuration for stanza
INFO      creating instance dumps directory: /pgdata/backup/dumps/16-main
```

La commande `pglift instance list` permet de lister les instances :

```
$ pglift instance list
+-----+-----+-----+-----+-----+
| name | version | port | datadir | status |
+-----+-----+-----+-----+-----+
| main | 16      | 5432 | /pgdata/16/main/data | running |
+-----+-----+-----+-----+-----+
```

Pour faciliter la connexion à l'instance, ou même l'administration des composants satellites qui lui sont associés, il est très utile de charger les variables d'environnement de l'instance. La commande `pglift instance env main` permet de les afficher :

```
$ pglift instance env main
PATH=/usr/pgsql-16/bin:/home/postgres/.local/bin:/home/postgres/bin:
>/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/opt/pglift/bin:/usr/pgsql-16/bin
PGBACKREST_CONFIG_PATH=/home/postgres/.local/share/pglift/etc/pgbackrest
PGBACKREST_STANZA=main
PGDATA=/pgdata/16/main/data
PGHOST=/var/run/postgresql
PGPASSFILE=/home/postgres/.pgpass
PGPORT=5432
PGUSER=postgres
```



```
PSQLRC=/pgdata/16/main/data/.psqlrc  
PSQL_HISTORY=/pgdata/16/main/data/.psql_history
```

Exporter le résultat de cette commande permet de charger les variables d'environnement de l'instance `main` sur la session courante.

```
$ export $(pglift instance env main)
```

Il est également possible de démarrer un nouveau *shell* avec les variables d'environnement de l'instance exportées :

```
$ pglift instance shell main
```

L'instance est alors accessible via `psql` sans option :

```
$ psql  
psql (16.4)  
Type "help" for help.
```

```
[16/main] postgres@~=#
```



Certaines commandes utilisées dans la suite de cet atelier nécessiteront que l'environnement de l'instance `main` soit chargé dans la session.

3/ Installation et création d'instance avec pglift (Ansible).

3.1 INVENTAIRE



Inventaire *Ansible* : ~/ansible/inventory

```
srv-pg1      ansible_port=2201
srv-pg2      ansible_port=2202
srv-pg3      ansible_port=2203
```

```
[all:vars]
ansible_connection=ssh
ansible_host=127.0.0.1
ansible_user=dalibo
ansible_ssh_pass=Password
```

```
[etcd]
srv-pg1
srv-pg2
srv-pg3
```

```
[database]
srv-pg1
srv-pg2
```

```
[primary]
srv-pg1
```

```
[standby]
srv-pg2
```

```
[temboard]
srv-pg3
```

3.2 COLLECTION



```
collections:
- dalibo.pglift
- community.general
- dalibo.essential
- dalibo.advanced
- dalibo.extras
```

```
$ ansible-galaxy collection install -fr collections/requirements.yml
```

Installer les collections requises pour déployer et manipuler *pglift* à travers *Ansible*, avec *ansible-galaxy*:

3.3 GÉNÉRATION DES CERTIFICATS TLS



\$ ansible-playbook -i inventory prerequisites.yml

Générer les certificats TLS avec le playbook suivant :

```
$ ansible-playbook -i inventory prerequisites.yml
```

3.4 INSTALLATION DE TEMBOARD UI



- Roles :
 - `dalibo.essential.temboard` : Pour l'installation
 - `temboard` : Pour la configuration
- Playbook: `temboard.yml`

```
$ ansible-playbook -i inventory temboard.yml
```

Exécuter le playbook `temboard.yml` sur l'inventaire établi afin de déployer l'interface `temBoard`:

```
$ ansible-playbook -i inventory temboard.yml
```

3.5 INSTALLATION DU CLUSTER ETCD



- Roles :
 - `dalibo.extras.etcd` : Pour l'installation et la configuration
- Playbook: `etcd.yml`

```
$ ansible-playbook -i inventory etcd.yml
```

Exécuter le playbook `etcd.yml` sur l'inventaire établi afin de déployer le cluster ETCD :

```
$ ansible-playbook -i inventory etcd.yml
```

3.6 INSTALLATION DE L'ENVIRONNEMENT



```
- name: Install Database Server
  hosts: database
  become: true
  roles:
    - dalibo.extras.repo_epel
    - dalibo.essential.repo_pgdg
    - dalibo.essential.repo_dalibo
    - dalibo.extras.accounts
    - dalibo.essential.postgresql
    - dalibo.essential.pgbackrest
    - dalibo.advanced.patroni
    - dalibo.essential.temboard_agent
    - dalibo.essential.pglift

$ ansible-playbook -i inventory postgresql.yml
```

Installer l'ensemble des éléments nécessaires au fonctionnement de *pglift*, de PostgreSQL et des outils satellites avec le playbook `postgresql.yml`

3.7 DÉPLOIEMENT D'UNE INSTANCE AVEC ANSIBLE



- Module `dalibo.pglift.instance`

L'instance peut être déployée par *Ansible*, via le module `dalibo.pglift.instance`. Pour cela, les serveurs doivent être accessibles depuis le poste local sur un compte utilisateur sudoer. Le compte utilisateur `dalibo` est utilisé à cet effet sur les machines de l'atelier.

3.8 CRÉATION DE L'INSTANCE



```
---  
  
- name: Deploy a standalone Instance  
  hosts: primary  
  become: true  
  become_user: postgres  
  tasks:  
    - name: Create Instance  
      dalibo.pglift.instance:  
        name: main  
        state: started  
        version: 16  
        port: 5432  
        surole_password: Passw0rd  
        pgbackrest:  
          password: Passw0rd  
          stanza: main-stz  
        temboard:  
          password: Passw0rd  
          replrole_password: Passw0rd  
        databases:  
          - name: ws1  
  
$ ansible-playbook -i inventory instance_standalone.yml
```

Exécuter le *playbook* `instance_standalone.yml` sur l'inventaire établi :

```
$ ansible-playbook -i inventory \  
instance_standalone.yml
```

3.8.1 Création d'un secondaire en streaming répllication



```

---

- name: Get instances gather facts
  hosts: primary
  gather_facts: true

- name: Deploy standby Instance
  hosts: standby
  become: true
  become_user: postgres
  tasks:
    - name: Creating standby Instance
      dalibo.pglift.instance:
        name: standby
        state: started
        version: 16
        port: 5432
        surole_password: Passw0rd
        pgbackrest:
          password: Passw0rd
          stanza: main-stz
        standby:
          primary_conninfo: "host={{ hostvars[item]['ansible_eth1'].ipv4.address }} user=replication p
            password: Passw0rd
          temboard:
            password: Passw0rd
          loop: "{{ groups['primary'] }}"

$ ansible-playbook -i inventory standby_instance.yml

```

Exécuter le *playbook* `standby_instance.yml` sur l'inventaire établi :

```
$ ansible-playbook -i inventory standby_instance.yml
```

4/ Sauvegarde et Restauration pgbackrest avec pglift

4.1 RAPPEL POINT IN TIME RECOVERY (PITR)



- Point In Time Recovery
- À chaud
- En continu
- Cohérente

PITR est l'acronyme de Point In Time Recovery, autrement dit restauration à un point dans le temps.

C'est une sauvegarde à chaud et surtout en continu. Là où une sauvegarde logique du type `pg_dump` se fait au mieux une fois toutes les 24 h, la sauvegarde PITR se fait en continu grâce à l'archivage des journaux de transactions. De ce fait, ce type de sauvegarde diminue très fortement la fenêtre de perte de données.

Bien que la sauvegarde se fasse à chaud, le rejeu des journaux de transactions après sa restauration permet de retrouver un état cohérent.

4.2 SAUVEGARDE



```
$ pglift instance backup main
INFO      backing up instance 16/main with pgBackRest
```

Initialiser une base de données ws1:

```
$ pglift database -i main create ws1
INFO      creating 'ws1' database in 16/main
```

Effectuer une première sauvegarde de l'instance :

```
$ pglift instance backup main
INFO      backing up instance 16/main with pgBackRest
```

Lister les sauvegardes disponibles :

```
$ pglift instance backups main
Available backups for instance 16/main
```

label	size	repo_size	date_start	date_stop	type	databases
20240118-084757F	30.6 MB	4.1 MB	2024-01-18 08:47:57+00:00	2024-01-18 08:48:03+00:00	full	postgres, ws1

Peupler la base de données ws1 à l'aide de pgbench :

```
$ /usr/pgsql-16/bin/pgbench -i ws1
dropping old tables...
NOTICE: table "pgbench_accounts" does not exist, skipping
NOTICE: table "pgbench_branches" does not exist, skipping
NOTICE: table "pgbench_history" does not exist, skipping
NOTICE: table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
100000 of 100000 tuples (100%) done (elapsed 0.07 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 0.18 s (drop tables 0.00 s, create tables 0.01 s, client-side generate 0.10
↪ s, vacuum 0.02 s, primary keys 0.05 s).
```

Effectuer une nouvelle sauvegarde de l'instance :

```
$ pglift instance backup main
INFO      backing up instance 16/main with pgBackRest
```

Lister à nouveau les sauvegardes disponibles :

```
$ pglift instance backups main
Available backups for instance 16/main
+-----+-----+-----+-----+-----+-----+-----+
| label | size | repo_size | date_start | date_stop | type | databases |
+-----+-----+-----+-----+-----+-----+-----+
| 20240118-084757F_20240118-085536I | 46.5 MB | 4.9 MB | 2024-01-18 08:55:36+00:00 | 2024-01-18 08:55:38+00:00 | incr | postgres, wsl |
| 20240118-084757F | 30.6 MB | 4.1 MB | 2024-01-18 08:47:57+00:00 | 2024-01-18 08:48:03+00:00 | full | postgres, wsl |
+-----+-----+-----+-----+-----+-----+-----+
```

Comme on peut le voir dans la liste de sauvegardes, la seconde est de type incrémentielle.

4.3 RESTAURATION



```
$ pglift database -i main drop ws1
INFO      dropping 'ws1' database
$ pglift instance stop main
INFO      stopping PostgreSQL 16-main
$ pglift instance restore main --label
↪ '20240118-084757F_20240118-085536I'
INFO      restoring instance 16/main with pgBackRest
```

Supprimer la base de données ws1:

```
$ pglift database -i main drop ws1
INFO      dropping 'ws1' database
```

Lister les bases de données :

```
$ pglift database list main
```

name	owner	encod.	col.	ctype	acls	size	description	tablespace
postgres	postgres	UTF8	C	C		7.6 MB	default administrative connection database	name: pg_default location: size: 37.9 MB
template1	postgres	UTF8	C	C	=c/postgres, postgres=Ctc/postgres	7.6 MB	default template for new databases	name: pg_default location: size: 37.9 MB

La base de données ws1 a bien été supprimée.

Pour restaurer une instance, il est nécessaire de l'arrêter :

```
$ pglift instance stop main
INFO      stopping PostgreSQL 16-main
```

Restaurer la sauvegarde effectuée avant la suppression de la base ws1 :

```
$ pglift instance restore main --label '20240118-084757F_20240118-085536I'
INFO      restoring instance 16/main with pgBackRest
```

Démarrer l'instance et lister les tables de la base ws1:

```
$ pglift instance start main
INFO      starting PostgreSQL 16-main
```

```
$ psql ws1
psql (16.6)
Type "help" for help.
```

```
ws1=# \dt
```

```
          List of relations
 Schema |      Name      | Type | Owner
-----+-----+-----+-----
 public | pgbench_accounts | table | postgres
 public | pgbench_branches | table | postgres
 public | pgbench_history  | table | postgres
 public | pgbench_tellers  | table | postgres
(4 rows)
```

L'instance a bien été restaurée avec la base de données ws1 après l'import de données par pgbench.

5/ Manipulation d'instances avec pglift

5.1 LISTER LES INSTANCES



`$ pglift instance list`

La commande `pglift instance list` permet de lister l'ensemble des instances managées par *pglift* sur le serveur.

`$ pglift instance list`

name	version	port	datadir	status
main	16	5432	/pgdata/16/main/data	running

5.2 OBTENIR LA DESCRIPTIONS DES INSTANCES



```
$ pglift instance get main
```

La commande `pglift instance get` permet d'obtenir la description d'une instance. On peut notamment y voir les locales, l'encodage, si un redémarrage est nécessaire pour prendre en compte une modification de configuration ou encore la présence de *slots* de réplication.

```
$ pglift instance get main
name    version  port  data_checksums  locale  encoding  pending_restart  replication_slots
main    16        5432  False           C       UTF8      False
```

5.3 OBTENIR LE STATUT D'UNE INSTANCE



`$ pglift instance status main`

La commande `pglift instance status` permet d'obtenir le statut d'une instance ainsi que des composants satellites qui sont utilisés par le socle (*Temboard*, *postgres_exporter*, *patroni*, etc) lorsque ceux-ci sont activés dans la configuration.

```
$ pglift instance status main
PostgreSQL: not running
```

5.4 GESTION DE L'ÉTAT



```
$ pglift instance start main
$ pglift instance stop main
$ pglift instance restart main
$ pglift instance reload main
```

La commande `pglift instance start` permet de démarrer l'instance via le service.

```
$ pglift instance start main
INFO      starting PostgreSQL cluster 16/main
```

La commande `pglift instance stop` permet de stopper l'instance via le service.

```
$ pglift instance stop main
INFO      stopping PostgreSQL cluster 16/main
```

Les deux actions peuvent être combinées pour effectuer un redémarrage avec la commande `pglift instance restart`.

```
$ pglift instance restart main
INFO      restarting instance 16/main
```

La commande `pglift instance reload` permet de recharger la configuration de l'instance.

```
$ pglift instance reload main
INFO      reloading PostgreSQL configuration for 16/main
```

5.5 CONSULTER LES TRACES



`$ pglift instance logs main`

Pour consulter les traces d'une instance, qui se trouvent dans le répertoire `/pgdata/log/postgresql`, il est possible de passer par la commande `pglift` plutôt que d'ouvrir manuellement le fichier.

`$ pglift instance logs main`

5.6 GESTION DE LA CONFIGURATION DE POSTGRESQL



- pglift peut manipuler la configuration des instances PostgreSQL avec la commande `pglift pgconf`
- Opérations possibles : `edit`, `remove`, `set`, `show`

La configuration PostgreSQL des instances créées par pglift est construite à partir de plusieurs sources :

- le fichier `postgresql.conf` présent dans le répertoire de données de l'instance.
- le fichier `template postgresql.conf` de la configuration de site.
- Tous les éléments de configuration que les composants satellites pourraient définir (par exemple, pgBackRest définirait `archive_command`, etc).
- Les éléments de configuration fournis par l'utilisateur.

Ce processus s'applique lors de la création de l'instance, mais aussi à chaque fois que l'instance est mise à jour via pglift.

pglift peut manipuler la configuration des instances PostgreSQL avec la commande `pglift pgconf`.

Les options sont :

```
$ pglift pgconf --help
Usage: pglift pgconf [OPTIONS] COMMAND [ARGS]...
```

Manage configuration of a PostgreSQL instance.

Options:

```
-i, --instance <version>/<name>
                                Instance identifier; the <version>/ prefix
                                may be omitted if there's only one instance
                                matching <name>. Required if there is more
                                than one instance on system.

--help                          Show this message and exit.
```

Commands:

```
edit      Edit managed configuration.
remove    Remove configuration items.
set       Set configuration items.
show      Show configuration (all parameters or specified ones).
```

5.6.1 Modifier la configuration



```
$ pglift pgconf -i main edit
$ pglift pgconf -i main set
```

La commande `pglift pgconf edit` permet de modifier directement le fichier de configuration en ouvrant un éditeur de texte avec le contenu du `postgresql.conf`.

Pour modifier spécifiquement certains paramètres, il est également possible de passer par la commande `pglift pgconf set`. Il est possible de lui passer un ou plusieurs paramètres sous la forme `paramètre=valeur`.

```
$ pglift pgconf -i main set max_connections=200 \
log_connections=on
```

```
INFO      configuring PostgreSQL
INFO      instance 16/main needs reload due to parameter changes: log_connections
INFO      reloading PostgreSQL configuration for 16/main
WARNING   instance 16/main needs restart due to parameter changes: max_connections
> PostgreSQL needs to be restarted; restart now? [y/n] (n): y
log_connections: off -> on
max_connections: 100 -> 200
```

5.6.2 Afficher la configuration



```
$ pglift pgconf -i main show
```

La commande `pglift pgconf show` permet d'afficher la configuration de l'instance.

```
$ pglift pgconf -i main show max_connections
max_connections = 200
```

5.6.3 Supprimer une configuration



```
$ pglift pgconf -i main remove
```

La suppression de paramètre est réalisée avec la commande `pglift pgconf remove`.

```
$ pglift pgconf -i main remove max_connections
INFO      configuring PostgreSQL
WARNING   instance 16/main needs restart due to parameter changes: max_connections
> PostgreSQL needs to be restarted; restart now? [y/n] (n): y
max_connections: 300 -> None
```

5.7 GESTION DU PG_HBA.CONF



- pglift peut manipuler le `pg_hba.conf` avec la commande `pglift pghba`
- Opérations possibles : add, remove

La commande `pglift pghba` permet de gérer les entrées du fichier `pg_hba.conf` d'une instance PostgreSQL.

Pour ajouter une entrée :

```
$ pglift pghba add --connection-type host --database all \  
--user all --connection-address 192.168.10.100/32 --method scram-sha-256  
INFO      entry added to pg_hba.conf
```

Pour supprimer une entrée :

```
$ pglift pghba remove --connection-type host --database all --user all \  
--connection-address 192.168.10.100/32 --method scram-sha-256  
INFO      entry removed from pg_hba.conf
```

5.8 MAINTENANCE DES DONNÉES



```
$ pglift database -i main run "ANALYZE"  
$ pglift database -i main run -d postgres "VACUUM FULL"  
$ pglift database -i main run -x postgres "VACUUM"
```

Les opérations de maintenance sur les données sont réalisées grâce à la commande `pglift database run`. Celle-ci permet d'exécuter une commande SQL sur une base de données spécifiée. L'instance cible doit également être précisée.

5.9 OPÉRATIONS SUR LES ROLES



```
$ pglift role

tasks:
  - name: my role
    dalibo.pglift.role:
      instance: main
      name: dba
      pgpass: true
      login: true
      connection_limit: 10
      valid_until: '2025-01-01T00:00'
      memberships:
        - role: pg_read_all_stats
```

La manipulation des rôles avec pglift est faite avec la commande `pglift role`.

Voici les options :

```
$ pglift role --help
```

```
Usage: pglift role [OPTIONS] COMMAND [ARGS]...
```

Manage roles.

Options:

```
-i, --instance <version>/<name>
```

Instance identifier; the <version>/ prefix may be omitted if there's only one instance matching <name>. Required if there is more than one instance on system.

```
--schema
```

Print the JSON schema of role model and exit.

```
--help
```

Show this message and exit.

Commands:

```
alter      Alter a role in a PostgreSQL instance
create     Create a role in a PostgreSQL instance
drop       Drop a role
get        Get the description of a role
list       List roles in instance
privileges List privileges of a role.
```

`set-profile` Set profile (read-only, read-write) for a specific role...

Pour créer une rôle :

```
$ pglift role create bob --login --replication --password
```

Pour modifier un rôle :

```
$ pglift role alter bob --superuser
```

Pour supprimer un rôle :

```
$ pglift role drop bob
```

Pour lister les rôles :

```
$ pglift role list
```

Il est également possible de créer, modifier et supprimer un rôle avec le module `dalibo.pglift.role` de la collection Ansible `dalibo.pglift`.

Voici un exemple :

```
tasks:
  - name: my role
    dalibo.pglift.role:
      instance: main
      name: dba
      pgpass: true
      login: true
      connection_limit: 10
      valid_until: '2025-01-01T00:00'
      memberships:
        - role: pg_read_all_stats
```

Cette tâche va créer un rôle `dba` sur l'instance `main`. Le mot de passe de ce rôle sera stocké dans le `pgpass` et il aura l'attribut `LOGIN`. Une limite de connexion et une date de validité pour le mot de passe sont également définies. Ce rôle appartiendra au groupe `pg_read_all_stats`.

5.10 OPÉRATIONS SUR LES BASES DE DONNÉES



```
$ pglift database

tasks:
- name: my database
  dalibo.pglift.database:
    instance: main
    name: myapp
    owner: dba
```

La manipulation des bases de données avec pglift est faite avec la commande `pglift database`.

Voici les options :

```
$ pglift database --help
```

```
Usage: pglift database [OPTIONS] COMMAND [ARGS]...
```

Manage databases.

Options:

<code>-i, --instance <version>/<name></code>	Instance identifier; the <code><version>/</code> prefix may be omitted if there's only one instance matching <code><name></code> . Required if there is more than one instance on system.
<code>--schema</code>	Print the JSON schema of database model and exit.
<code>--help</code>	Show this message and exit.

Commands:

<code>alter</code>	Alter a database in a PostgreSQL instance
<code>create</code>	Create a database in a PostgreSQL instance
<code>drop</code>	Drop a database
<code>dump</code>	Dump a database
<code>dumps</code>	List the database dumps
<code>get</code>	Get the description of a database
<code>list</code>	List databases (all or specified ones)
<code>privileges</code>	List privileges on a database.
<code>restore</code>	Restore a database dump
<code>run</code>	Run given command on databases of a PostgreSQL instance

Pour créer une base de données :

```
$ pglift database create db01 --schema s1 --owner bob
```

Pour modifier une base de données :

```
$ pglift database alter db01 --owner alice
```

Pour supprimer une base de données :

```
$ pglift database drop db01
```

Pour lister les base de données :

```
$ pglift database list
```

Il est également possible de créer, modifier et supprimer une base de données avec le module `dalibo.pglift.database` de la collection Ansible `dalibo.pglift`.

Voici un exemple :

```
tasks:
- name: my database
  dalibo.pglift.database:
    instance: main
    name: myapp
    owner: dba
```

Cette tâche va créer une base de données `myapp` sur l'instance `main` et le propriétaire de cette base de données sera `dba`.

5.11 RÉPLICATION



```
$ pglift instance create <INSTANCE> --standby-for
tasks:
- name: Creating standby Instance
  dalibo.pglift.instance:
    name: standby
    [...]
    standby:
      primary_conninfo: "host=X.X.X.X user=replication port=5432"
      password: Passw0rd
```

La création d'une instance répliquée est réalisée avec la commande `pglift instance create` et l'option `--standby-for`.

Avant de créer l'instance secondaire, il est nécessaire de s'assurer que l'instance primaire autorise les connexions de réplication depuis le serveur hébergeant l'instance secondaire.

Pour créer l'instance secondaire, exécuter la commande suivante :

```
$ pglift instance create standby \
--standby-for 'host=X.X.X.X user=replication port=5432' --standby-password Passw0rd
```

Il est également possible de passer par le module `dalibo.pglift.instance` de la collection Ansible `dalibo.pglift`.

```
tasks:
- name: Creating standby Instance
  dalibo.pglift.instance:
    name: standby
    state: started
    version: 16
    port: 5433
    surole_password: Passw0rd
    replrole_password: Passw0rd
    pgbackrest:
      password: Passw0rd
      stanza: main-stz
    standby:
      primary_conninfo: "host=X.X.X.X user=replication port=5432"
      password: Passw0rd
```

Les paramètres `standby.primary_conninfo` et `standby.password` permettent de définir respectivement la chaîne de connexion au primaire et le mot de passe de connexion.

6/ Déployer des instances en Haute Disponibilité avec pglift

6.1 CONFIGURATION DE PGLIFT



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

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

```
---
patroni:
  execpath: /usr/bin/patroni
  configpath: /etc/patroni/{name}.yaml
  logpath: /var/log/patroni
  etcd:
    hosts:
      - <etcd1>:2379
      - <etcd2>:2379
      - <etcd3>:2379
    protocol: https
    cacert: "/etc/pki/tls/certs/ca_cert.pem"
    cert: "/etc/pki/tls/certs/<hostname>.pem"
    key: "/etc/pki/tls/private/<hostname>.key"
  restapi:
    cafile: "/etc/pki/tls/certs/ca_cert.pem"
    certfile: "/etc/pki/tls/certs/<hostname>.pem"
    keyfile: "/etc/pki/tls/private/<hostname>.key"
  ctl:
    certfile: "/etc/pki/tls/certs/<hostname>.pem"
    keyfile: "/etc/pki/tls/private/<hostname>.key"
  postgresql:
    use_pg_rewind: true
    passfile: /var/lib/pgsql/{name}.pgpass
  watchdog:
    mode: required
    device: /dev/watchdog
    safety_margin: 5
```

En ajoutant cette section dans la configuration, les options liés aux noeuds *patroni* sont présents

lors de la commande `pglift instance create` (voir `pglift instance create --help`). Il sera également possible de déployer les noeuds à l'aide du module Ansible `dalibo.pglift.instance`.

Pour prendre en compte la modification de la configuration de `pglift`, il est nécessaire de désinstaller puis de réinstaller la configuration de site :

```
$ pglift site-configure uninstall  
$ pglift site-configure install
```

6.2 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 alors avec un *playbook Ansible*.

6.3 CLI



```
$ pglift instance create main --pgbackrest-stanza=main-app \  
  --replrole-password Passw0rd \  
  --patroni-cluster maincluster \  
  --patroni-node <node_name> \  
  --patroni-restapi-connect-address "<IP-Machine>:8008" \  
  --patroni-restapi-listen "<IP-Machine>:8008" \  
  --patroni-postgresql-connect-host <IP-Machine>
```

6.4 ANSIBLE



- Module `dalibo.pglift.instance`

Un noeud `patroni` peut être déployée par *Ansible*, via le module `dalibo.pglift.instance` et les paramètres liés à `patroni`.

6.5 CRÉATION DES NOEUDS DU CLUSTER



```
---
- name: Create Instances
  hosts: database
  become_user: postgres
  become: true
  tasks:

    - name: Managing instances
      dalibo.pglift.instance:
        name: main
        state: started
        version: 16
        port: 5432
        surole_password: Passw0rd
        replrole_password: Passw0rd
        temboard:
          password: Passw0rd
        pgbackrest:
          password: Passw0rd
          stanza: maincluster-stz
        patroni:
          cluster: maincluster
          node: "{{ ansible_hostname }}"
          restapi:
            connect_address: "{{ ansible_eth1.ipv4.address }}:8008"
            listen: "{{ ansible_eth1.ipv4.address }}:8008"
          postgresql:
            connect_host: "{{ ansible_default_ipv4.address }}"
        throttle: 1

$ ansible-playbook -i inventory instance_ha.yml
```

Exécuter le *playbook* `instance_ha.yml` sur l'inventaire établi :

```
$ ansible-playbook -i inventory instance_ha.yml
```

6.6 GESTION DE LA CONFIGURATION DU CLUSTER

6.6.1 Avec Patroni



```
$ pglift instance exec main -- patronictl edit-config maincluster
```

La configuration modifiée avec *patroni* est appliquée à l'ensemble du cluster. Il est possible de changer la configuration *patroni* ainsi que la configuration PostgreSQL.

```
$ pglift instance exec main -- patronictl edit-config maincluster
```

6.6.2 Avec pglift



```
$ pglift pgconf -i main edit
```

La configuration modifiée avec *pglift* est appliquée uniquement à l'instance du nœud PostgreSQL.

```
$ pglift pgconf -i main edit
```

6.7 STATUT DU CLUSTER



```
$ 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 |
+-----+-----+-----+-----+-----+-----+
↪ +
```

Il est possible d'afficher la topologie du cluster avec la commande `patronictl list` :

```
$ 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 |
+-----+-----+-----+-----+-----+-----+
```

- La valeur `Leader` dans la colonne rôle indique l'instance primaire.
- La valeur `Sync standby` dans la colonne rôle indique l'instance secondaire synchrone si le mode synchrone de *patroni* est actif.
- La colonne `State` donne l'état du nœud.
- La colonne `TL` indique la timeline des journaux de transactions de l'instance. En fonctionnement nominal, elle doit être identique sur tous les nœuds.
- La colonne `Lag` donne le retard de réplication du nœud par rapport à son instance source (l'instance primaire).

6.8 SWITCHOVER



```
$ pglift instance exec main -- patronictl switchover
```

Pour effectuer un *switchover* immédiat et de force, depuis n'importe quel nœud du cluster, avec la commande `patronictl` :

```
$ pglift instance exec main -- patronictl switchover
Current cluster topology
+ 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  | Replica | streaming | 1  |         0 |
+-----+-----+-----+-----+-----+-----+
Primary [srv-pg1]:
Candidate ['srv-pg2'] []: srv-pg2
When should the switchover take place (e.g. 2024-02-08T11:45 ) [now]: now
Are you sure you want to switchover cluster maincluster, demoting current leader
↪ srv-pg1? [y/N]: y
2024-02-12 20:45:18.84531 Successfully switched over to "srv-pg2"
+ Cluster: maincluster (7334815726709754190) -----+-----+-----+
| Member | Host           | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| srv-pg1 | 192.168.121.172 | Replica | stopped |    | unknown    |
| srv-pg2 | 192.168.121.89  | Leader | running | 1  |           |
+-----+-----+-----+-----+-----+-----+
```

Vérifier que le deuxième nœud est bien “Leader”:

```
$ pglift instance exec main -- patronictl list
+ Cluster: maincluster (7334815726709754190) -----+-----+-----+
| Member | Host           | Role   | State   | TL | Lag in MB |
+-----+-----+-----+-----+-----+-----+
| srv-pg1 | 192.168.121.172 | Replica | streaming | 2  |         0 |
| srv-pg2 | 192.168.121.89  | Leader | running | 2  |           |
+-----+-----+-----+-----+-----+-----+
```

6.9 FAILOVER



- Simuler une panne en arrêtant le processus *patroni* :

```
$ pkill -9 patroni
```

- *srv-pg2* redémarre suite au déclenchement du *watchdog* par *patroni*

Pour simuler une panne, le processus *patroni* va être stoppé avec la commande `pkill -9` sur *srv-pg2* :

```
$ pkill -9 patroni
```

Depuis *srv-pg1*, vérifier l'état du cluster :

```
$ 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 | 3 |           |
| srv-pg2 | 192.168.121.89  | Replica | running | 3 |          0 |
+-----+-----+-----+-----+-----+-----+

```

Il est possible alors de voir que *srv-pg1* est promu Leader et ouvert aux écritures.

Le *watchdog* étant configuré, *srv-pg2* redémarre après avoir utilisé la commande `pkill` et *patroni* se charge de reconstruire le primaire en serveur secondaire.

7/ Gestion de parc d'instances avec temBoard.

7.1 PRÉSENTATION DE TEMBOARD



- *temBoard* : supervision et administration de PostgreSQL
- Un composant serveur, paquet `temboard`
- Un agent pour les serveurs PostgreSQL, paquet `temboard-agent`

temBoard est un outil de supervision et d'administration de parc d'instances PostgreSQL. Il est constitué d'un composant serveur et d'un agent conçu pour fonctionner sur les serveurs PostgreSQL.

Dans cet atelier, un serveur *temBoard* fonctionnera sur le serveur `srv-helper`. Une instance supervisée et un agent *temBoard* associé seront déployés sur `srv-pg1` à l'aide de *pglift*.

7.2 ENREGISTREMENT DE L'INSTANCE



```
---

- name: Get instances gather facts
  hosts: database
  gather_facts: true

- name: Register instances in temBoard
  hosts: temboard
  gather_facts: true
  become: true
  become_user: temboard
  tasks:
    - name: Register instances in temBoard
      ansible.builtin.shell: "temboard register-
instance {{ hostvars[item]['ansible_default_ipv4']['address']} } 2345 -e default"
      loop: "{{ groups['primary'] }}"

    - name: Register instances in temBoard
      ansible.builtin.shell: "temboard register-
instance {{ hostvars[item]['ansible_default_ipv4']['address']} } 2345 -e default"
      loop: "{{ groups['standby'] }}"

$ ansible-playbook -i inventory temboard_register.yml
```

Exécuter le playbook `temboard_register.yml` afin de réaliser l'enregistrement de l'instance primaire et secondaire dans l'interface temBoard:

```
$ ansible-playbook -i inventory temboard_register.yml
```

7.3 VISUALISATION DES MÉTRIQUES



- Vue *Dashboard* : Résumé des métriques
- Vue *Activity* : Activité de l'instance en temps réel
- Vue *Monitoring* : Afficher des métriques sous forme de graphes
- Vue *Status* : Statut des sondes de supervision
- Vue *Maintenance* :
 - Informations sur l'utilisation des disques
 - Possibilité de déclencher des opérations de maintenance (VACUUM [FULL], ANALYZE, REINDEX)
- Vue *Configuration* : Modifier le paramétrage de l'instance

L'interface web de *temBoard* présente un certain nombre d'indicateurs sur l'instance enregistrée disponible depuis : <https://srv-helper:8888>

La vue *Dashboard* offre une vue d'ensemble des métriques de supervision de l'instance. On y retrouve les informations provenant de sondes qui surveillent les éléments suivants :

- utilisation du CPU, de la mémoire, du disque
- utilisation du cache disque (*cache hit ratio*)
- nombre de connexions établies, session en attente
- nombre de transactions par secondes (*tps*)
- génération de fichiers WAL
- génération de fichiers temporaires
- fragmentation (*bloat*) dans les tables et index.

La vue *Activity* permet de visualiser l'activité de l'instance en temps réel, notamment obtenir des informations sur les processus en cours, leur état, et les ordres SQL qu'ils exécutent. Il est possible de filtrer les sessions bloquantes (*blocking*) ou en attente (*waiting*) via les onglets respectifs.

La vue *Monitoring* permet de choisir des métriques à afficher sous forme de graphe, sur une plage de temps personnalisable.

La vue *Status* liste l'ensemble des sondes avec leur état actuel. Cliquer sur une sonde permet d'afficher le graphe correspondant.

La vue *Maintenance* établit une liste des bases de données de l'instance, avec des informations sur

l'occupation d'espace disque et les taux de fragmentation associés. Les informations sur l'utilisation du disque sont alors détaillées par schéma, et des boutons apparaissent en haut de la page pour déclencher des opérations courantes de maintenance :

- VACUUM
- VACUUM FULL
- ANALYZE
- REINDEX

La vue *Configuration* permet la configuration de l'instance depuis *temBoard*. Pour trouver un paramètre, il est possible de le rechercher ou bien de le retrouver dans les différentes catégories.

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/migorpg>
- 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.

