et sans LDAP

GRANT et REVOKE avec ldap2pg

DALIBO
L'expertise PostgreSQL

Parlons de vos données -- 22 février 2018

@EtienneBersac

GRANT et REVOKE avec ldap2pg

et sans LDAP

TITRE : GRANT et REVOKE avec ldap2pg
SOUS-TITRE : et sans LDAP

DATE: Parlons de vos données – 22 février 2018

À PROPOS

- Étienne BERSAC
- Développeur chez Dalibo SCOP
- github.com/bersace

LDAP2PG

- github.com/dalibo/ldap2pg
- industrialisation et sécurisation des accès Postgres.
- cf. dali.bo/pgsession9-ldap2pg (~30m)

En juin 2017, Dalibo a lancé le développement d'un outil de synchronisation des rôles et des ACLs dans un cluster Postgres à partir d'un annuaire LDAP. Le projet nommé `ldap2pg` est disponible sur GitHub. De nombreuses versions ont été livrées depuis et une petite communauté d'utilisateurs s'est formé autour du projet.

`ldap2pg` est une clef dans le processus d'industrialisation de Postgres dans les grandes entreprises. Il reste néanmoins suffisamment simple pour être utile pour quelques instances avec peu de rôles. La limite, est que le mot de passe n'est pas géré à la création du rôle.

Le 22 novembre 2017, j'ai présenté `ldap2pg` à la PGSession 9 avec une démonstration de la création des rôles à partir d'un annuaire OpenLDAP. La vidéo est sur le YouTube Dalibo.

SYNCHRO AUTOMATIQUE

- réinitialiser (tout)
- redérouler les autorisations
- groupes ro, rw, ddl

Entrons dans le vif du sujet. On retrouve souvent le même scénario dans les solutions maison d'automatisation de la synchronisation des ACL : on révoque tout les droits et on réexécute les autorisations sur une feuille blanche.

Cela implique une rupture du service. En outre, le coût de synchronisation va croissant avec le nombre d'objets en base et le nombre de rôles à configurer. L'utilisation des groupes est une amélioration non négligeable, parfois ignorée.

SYNCHRO LDAP2PG

- introspection.
- autorisation explicite.
- révocation implicite.

`ldap2pg` fait un pari ambitieux : on peut interroger Postgres pour connaître les privilèges accordés pour ne révoquer que ce qui est accordé et n'accorder que ce qui ne l'est pas.

Avec l'introspection on peut aller vers un mode plus contrôllé : tout ce qui n'est pas accordé explicitement doit implicitement être révoqué. `ldap2pg` travaille en trois étapes successives : inspection des ACLs accordées, comparaison avec les ACLs demandées, révocations et autorisations pour atteindre la correspondance.

DÉFINITION D'UNE ACL

```
acls:
  connect:
    type: datacl  # ou nspacl, defacl, globaldefacl
    inspect: SELECT …
    grant: GRANT CONNECT ON … TO {role};
    revoke: REVOKE CONNECT ON … FROM {role};
  ro:
  - connect
```

Pour que `ldap2pg` puisse gérer une ACL, le DBA doit la décrire. C'est une partie très technique qui demande de bien connaître les entrailles de Postgres, parfois même le code C !

Comme dans Postgres, `ldap2pg` distingue différents types d'ACL. On reprends la même nomenclature. Suivant le type d'ACL, `ldap2pg` assiste la bouclage selon plusieurs axes : base de données, schémas, propriétaires ou une combinaison de tout ça. Ça peut devenir très gros.

Ensuite, `ldap2pg` laisse le DBA faire ce qu'il préfère : écrire des requêtes SQL :-) Trois requêtes définissent une ACL : une pour inspecter les ACLs accordées en base. Cette requête est exécutée sur chaque base gérée. Ensuite les requêtes grant et revoke, évidemment. Si la requêtes d'inspection n'est pas définie, `ldap2pg` ne saura pas révoquer, et réaccordera sans cesse l'ACL.

Un point important de l'introspection est la capacité de détecter si un `GRANT ON ALL TABLES` est effectivement accordé à toute les tables/vues d'un schéma. `ldap2pg` détecte qu'un `GRANT ON ALL TABLES` est incomplet, et réexécute la requêt `GRANT`.

Enfin, on peut grouper les ACL. Vu que l'introspection est complexe, c'est nécessaire de découper les ACL en ACL élémentaires. Petit défi : écrire le `SELECT` qui retourne tout les rôles qui ont `GRANT ALL PRIVILEGES ON ALL TABLES` ;-)

ACL PRÉDÉFINIES

- inclue la requête d'introspection.
- inactive par défaut.
- activable via un groupe actif.
- ldap2pg.rtfd.io/en/latest/wellknown

Vu la technicité de l'écriture de requête d'inspection, alors que c'est l'intérêt de `ldap2pg`, j'ai d'abord envisagé de simplement fournir des exemples dans la doc. Problème : c'est difficilement testable, ça risque de générer beaucoup de tickets. En plus, j'ai pas envie de recopier tout le temps les mêmes requêtes, un `ldap2pg.yml` doit rester le plus agréable possible à lire. J'ai finalement opté pour livrer avec `ldap2pg` des ACLs testées.

Le premier problème, c'est de ne pas déclencher la révocation implicite sur tout les ACL présent en base ! Solution : `ldap2pg` n'inspecte pas les ACLs commençant par `_`. Parcontre, si un groupe d'ACL **ne** commence **pas** par `_`, il est actif et toute les ACLs du groupes sont actives par transitivité. Donc on active une ACL en l'incluant dans un groupe actif.

Il y a déjà pas mal d'ACL prédéfinies, bien que ça ne soit pas exhaustif. Les usages courants sont couvert. Il manque par exemple les ACL pour les wrapper de données distantes, les extensions. Postgres restreint déjà l'accès à ces objets aux superutilisateurs, on pas encore eut besoin de gérer ça avec `ldap2pg`. Une page de documentation est générée à partir des ACLs prédéfinie dans la version `master`.

ALTER DEFAULT PRIVILEGES

- anticiper la modification du modèle.
- lister les propriétaires.
- par schéma ou global.
- … bug :-)

Je profite qu'on parle des ACL pour vous présenter `ALTER DEFAULT PRIVILEGES`. Je vois souvent des scripts qui sont lançés après la modification du schéma pour réaligner les droits sur les nouveaux objets. Une meilleue solution existe. `ALTER DEFAULT PRIVILEGES` signifie « accorde aux futurex tables ».

Dans `ldap2pg`, l'ACL `__select_on_tables__` est en fait un groupe accordant `SELECT` au tables existantes et aux tables futures, sans relancer `ldap2pg`. Le seul soucis est qu'il faut les appliquer à chaque rôles ayant des droits DDL. `ldap2pg` assiste le DBA en allignant les privilèges par défaut des propriétaires potentiels de nouveaux objets en base.

Par défaut, `ldap2pg` liste les superutilisateurs comme propriétaires gérés. C'est paramétrable.

En travaillant sur cette fonctionnalité, j'ai rencontré une erreur dans Postgres : la commande `ALTER DEFAULT PRIVILEGES REVOKE EXECUTE ON FUNCTIONS IN SCHEMA public FROM public;` n'a aucun effet. Cela signifie que vous ne pouvez pas révoquer le droit d'exécution des fonctions futures à un schéma particulier. `ldap2pg` inclus le contournement : révoquer globalement ce privilèges et à un schéma particulier. Petite note : le code de Postgres est très classe.

DÉMO !

Pour réinitialiser la démo: `make reset`

Étapes de la démo :

- pip3 install ldap2pg
- docker-compose up -d
- psql, `\du+`
- présenter fixture.sql
- rédiger un yaml
  - base et schémas: limiter à postgres, public, pg_catalog
  - owners_query
  - ACLs: ro, rw, admin
  - sync_map:
    - lecteur, ro
    - éditeur (lecteur), rw sur public
    - admin (éditeur), admin sur public, rw sur `pg_catalog`.
- exécuter et analyser la sortie. remarquer :
  - la synchronisation des rôles, olivier est détruit proprement.
  - la révocation des ACLs par défaut à `public`.
  - la décomposition en ACL élémentaires.
- tester:
  - psql -U admin, faire un CREATE dans pg_catalog, dans public
  - psql -U éditeur, faire un CREATE, un INSERT
  - psql -U lecteur, faire un INSERT, un SELECT

MERCI !

```
ldapsearch -b dc=pg,dc=meetup,dc=paris (objectClass=question)
```