LISV Laboratoire d'ingénierie des systèmes de Versailles
UVSQ université PARIS-SACLAY

Gestion d'un système de bases de données – TP : Installation et Configuration de PostgreSQL

Halim Djerroud

révision 1.0

Objectifs du TP

À l'issue de ce TP, vous serez capable de :
— Installer et configurer un serveur PostgreSQL sur Linux
— Comprendre l'architecture utilisateur PostgreSQL
— Créer et gérer des utilisateurs et des rôles
— Configurer le contrôle d'accès avec pg_hba.conf
— Mettre en place des règles de pare-feu
— Utiliser pgAdmin de manière sécurisée
— Appliquer le principe du moindre privilège avec les rôles
Durée : 2h30
Prérequis : Machine virtuelle Linux (Debian/Ubuntu) avec accès root

1 Partie 1 : Installation et Configuration Initiale (30 min)

1.1 Exercice 1.1 : Préparation du système

1. Mettez à jour votre système :

```
sudo apt update
sudo apt upgrade -y
```

2. Vérifiez l'espace disque disponible :

```
df -h
```

Question 1.1 : Combien d'espace est disponible sur la partition racine ?

3. Configurez le hostname de votre serveur :

```
sudo hostnamectl set-hostname postgres-tp
```

4. Vérifiez la configuration réseau :

```
ip addr show
hostname
```

Question 1.2 : Quelle est l'adresse IP de votre machine ?

1.2 Exercice 1.2 : Installation de PostgreSQL

1. Installez PostgreSQL :

```
sudo apt install postgresql postgresql-contrib -y
```

2. Vérifiez l'installation :

```
psql --version
```

Question 1.3 : Quelle version de PostgreSQL avez-vous installée ?

3. Vérifiez que le service est démarré :

```
sudo systemctl status postgresql
```

4. Si le service n'est pas actif, démarrez-le :

   ```
   sudo systemctl start postgresql
   ```

5. Activez le démarrage automatique :

   ```
   sudo systemctl enable postgresql
   ```

6. Vérifiez que PostgreSQL écoute bien :

   ```
   sudo ss -tlnp | grep postgres
   ```

   Question 1.4 : Sur quelle adresse IP et quel port PostgreSQL écoute-t-il par défaut ?

1.3 Exercice 1.3 : Première connexion et configuration

1. PostgreSQL crée automatiquement un utilisateur système `postgres`. Basculez sur cet utilisateur :

   ```
   sudo -i -u postgres
   ```

2. Lancez le client psql :

   ```
   psql
   ```

3. Une fois connecté, affichez les bases de données :

   ```
   \l
   ```

4. Affichez les utilisateurs/rôles :

   ```
   \du
   ```

5. Vérifiez l'utilisateur actuel :

   ```
   SELECT current_user;
   ```

6. Affichez des informations sur la connexion :

   ```
   \conninfo
   ```

7. Définissez un mot de passe pour l'utilisateur postgres :

   ```
   ALTER USER postgres WITH PASSWORD 'Postgres2024!Secure';
   ```

8. Quittez psql et retournez à votre utilisateur normal :

   ```
   \q
   exit
   ```

   Question 1.5 : Quelle est la différence entre un utilisateur système (postgres) et un utilisateur PostgreSQL ?

1.4 Exercice 1.4 : Exploration des fichiers de configuration

1. Localisez les fichiers de configuration (version 15, adapter selon votre version) :

   ```
   ls -la /etc/postgresql/15/main/
   ```

2. Affichez la configuration principale :

   ```
   sudo nano /etc/postgresql/15/main/postgresql.conf
   ```

   Note : Notez les paramètres importants (listen_addresses, port, max_connections).

3. Affichez le fichier de contrôle d'accès :

   ```
   sudo nano /etc/postgresql/15/main/pg_hba.conf
   ```

   Question 1.6 : Que signifie "HBA" dans pg_hba.conf ?

4. Localisez le répertoire des données :

   ```
   ls -la /var/lib/postgresql/15/main/
   ```

   Question 1.7 : Pourquoi les fichiers de données appartiennent-ils à l'utilisateur postgres ?

2 Partie 2 : Gestion des Utilisateurs et des Rôles (45 min)

2.1 Exercice 2.1 : Création de rôles et d'utilisateurs

Connectez-vous à PostgreSQL en tant que postgres :

```
sudo -u postgres psql
```

1. Créez un rôle administrateur :

   ```
   CREATE ROLE admin_db WITH LOGIN PASSWORD 'AdminDB2024!'
   CREATEDB CREATEROLE;
   ```

2. Créez un utilisateur pour une application web :

   ```
   CREATE USER appweb WITH PASSWORD 'WebApp2024!';
   ```

3. Créez un utilisateur en lecture seule :

   ```
   CREATE USER readonly WITH PASSWORD 'ReadOnly2024!';
   ```

4. Créez un rôle sans connexion (groupe) :

   ```
   CREATE ROLE lecteur NOLOGIN;
   ```

5. Listez tous les rôles créés :

   ```
   \du
   ```

   Question 2.1 : Quelle est la différence entre CREATE ROLE et CREATE USER ?

2.2 Exercice 2.2 : Création de bases de données

1. Créez une base de données pour une application e-commerce :

   ```
   CREATE DATABASE ecommerce
   OWNER appweb
   ENCODING 'UTF8'
   LC_COLLATE 'fr_FR.UTF-8'
   LC_CTYPE 'fr_FR.UTF-8';
   ```

2. Créez une base de données pour un blog :

   ```
   CREATE DATABASE blog
   OWNER admin_db
   ENCODING 'UTF8';
   ```

3. Créez une base de données pour les statistiques :

   ```
   CREATE DATABASE stats
   OWNER postgres
   ENCODING 'UTF8';
   ```

4. Listez toutes les bases de données :

   ```
   \l
   ```

   Question 2.2 : Pourquoi spécifier un propriétaire (OWNER) lors de la création d'une base ?

2.3 Exercice 2.3 : Attribution de privilèges

1. Accordez les droits de connexion sur la base ecommerce à appweb :

   ```
   GRANT CONNECT ON DATABASE ecommerce TO appweb;
   ```

2. Connectez-vous à la base ecommerce :

   ```
   \c ecommerce
   ```

3. Accordez tous les privilèges sur le schéma public à appweb :

   ```
   GRANT ALL PRIVILEGES ON SCHEMA public TO appweb;
   ```

4. Accordez les droits sur toutes les tables actuelles :

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA public TO appweb;
```

5. Important : accordez aussi les privilèges sur les futures tables :

```
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT, INSERT, UPDATE, DELETE
ON TABLES TO appweb;
```

6. Connectez-vous à la base stats :

```
\c stats
```

7. Accordez les droits de lecture à readonly :

```
GRANT CONNECT ON DATABASE stats TO readonly;
GRANT USAGE ON SCHEMA public TO readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
GRANT SELECT ON TABLES TO readonly;
```

Question 2.3 : Pourquoi faut-il utiliser ALTER DEFAULT PRIVILEGES ?

2.4 Exercice 2.4 : Utilisation des rôles comme groupes

1. Créez un rôle "éditeur" avec privilèges d'édition :

```
CREATE ROLE editeur NOLOGIN;
```

2. Connectez-vous à ecommerce et accordez des privilèges au rôle :

```
\c ecommerce
GRANT SELECT, INSERT, UPDATE, DELETE
ON ALL TABLES IN SCHEMA public TO editeur;
```

3. Accordez les privilèges au rôle lecteur :

```
GRANT SELECT ON ALL TABLES IN SCHEMA public TO lecteur;
```

4. Assignez des utilisateurs aux rôles :

```
-- appweb devient membre du groupe editeur
GRANT editeur TO appweb;

-- readonly devient membre du groupe lecteur
GRANT lecteur TO readonly;
```

5. Créez un nouvel utilisateur et assignez-le à un rôle :

```
CREATE USER analyste WITH PASSWORD 'Analyst2024!';
GRANT lecteur TO analyste;
GRANT CONNECT ON DATABASE stats TO analyste;
```

6. Vérifiez les appartenances :

```
\du
```

Question 2.4 : Quel est l'avantage d'utiliser des rôles comme groupes ?

2.5 Exercice 2.5 : Tests de connexion et privilèges

1. Quittez psql et testez la connexion avec appweb :

```
psql -h localhost -U appweb -d ecommerce
```

Note : Si la connexion échoue, c'est normal ! Nous allons configurer pg_hba.conf.

2. Retournez en tant que postgres :

```
sudo -u postgres psql
```

3. Vérifiez les privilèges d'appweb sur ecommerce :

```
\c ecommerce
\dp
```

Question 2.5 : La commande \dp affiche quoi ?

3 Partie 3 : Configuration Réseau et Sécurité (45 min)

3.1 Exercice 3.1 : Configuration du fichier pg_hba.conf

Le fichier pg_hba.conf contrôle qui peut se connecter à PostgreSQL et comment.

1. Éditez le fichier pg_hba.conf (adapter la version) :

```
sudo nano /etc/postgresql/15/main/pg_hba.conf
```

2. Observez les lignes existantes. Format :

```
# TYPE  DATABASE    USER        ADDRESS          METHOD
```

3. Ajoutez les lignes suivantes AVANT les lignes existantes :

```
# Connexions locales avec mot de passe
local   all         postgres                     scram-sha-256
local   all         all                          scram-sha-256

# Connexions depuis localhost
host    all         all         127.0.0.1/32     scram-sha-256
host    all         all         ::1/128          scram-sha-256

# Connexions depuis le réseau local (adapter à votre réseau)
host    all         all         192.168.0.0/16   scram-sha-256

# Accès spécifique pour appweb depuis serveur web
host    ecommerce   appweb      192.168.1.50/32 scram-sha-256

# Refuser tout le reste (optionnel, pour être explicite)
host    all         all         0.0.0.0/0        reject
```

Question 3.1 : Quelle est la différence entre "local" et "host" ?

4. Rechargez la configuration (sans redémarrage) :

```
sudo systemctl reload postgresql
```

5. Testez maintenant la connexion avec appweb :

```
psql -h localhost -U appweb -d ecommerce
```

6. Depuis psql, vérifiez la méthode d'authentification :

```
\conninfo
```

7. Quittez et testez avec readonly :

```
psql -h localhost -U readonly -d stats
```

Question 3.2 : Pourquoi utiliser scram-sha-256 plutôt que md5 ?

3.2 Exercice 3.2 : Configuration de l'écoute réseau

1. Éditez postgresql.conf :

```
sudo nano /etc/postgresql/15/main/postgresql.conf
```

2. Cherchez la ligne `listen_addresses` (vers ligne 59) :

```
#listen_addresses = 'localhost'
```

3. Décommentez et modifiez pour écouter sur toutes les interfaces :

```
listen_addresses = '*'
```

ATTENTION : En production, préférer une IP spécifique ou maintenir localhost avec tunnel SSH.

4. Vérifiez aussi le port :

```
port = 5432
```

5. Sauvegardez et redémarrez PostgreSQL :

```
sudo systemctl restart postgresql
```

6. Vérifiez que PostgreSQL écoute maintenant sur toutes les interfaces :

```
sudo ss -tlnp | grep 5432
```

Question 3.3 : Vous devriez voir 0.0.0.0 :5432. Que signifie cette adresse ?

3.3 Exercice 3.3 : Configuration du pare-feu

1. Vérifiez l'état du pare-feu :

```
sudo ufw status
```

2. Si le pare-feu n'est pas actif, activez-le :

```
sudo ufw enable
```

3. Autorisez SSH (important !) :

```
sudo ufw allow 22/tcp
```

4. Autorisez PostgreSQL uniquement depuis le réseau local :

```
sudo ufw allow from 192.168.0.0/16 to any port 5432
```

Note : Adaptez le réseau à votre configuration.

5. Bloquez tout le reste par défaut :

```
sudo ufw default deny incoming
sudo ufw default allow outgoing
```

6. Vérifiez les règles :

```
sudo ufw status verbose
```

7. Testez depuis une autre machine du réseau local (si disponible) :

```
# Depuis une autre machine
psql -h IP_DU_SERVEUR -U appweb -d ecommerce
```

Question 3.4 : Décrivez les 3 couches de sécurité mises en place.

3.4 Exercice 3.4 : Installation et configuration de pgAdmin

pgAdmin peut être installé en mode serveur (web) ou desktop. Nous allons installer la version web.

1. Ajoutez le dépôt pgAdmin :

```
curl -fsS https://www.pgadmin.org/static/packages_pgadmin_org.pub \
  | sudo gpg --dearmor -o /usr/share/keyrings/pgadmin-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/pgadmin-archive-keyring.gpg] \
https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/$(lsb_release -cs) \
pgadmin4 main" | sudo tee /etc/apt/sources.list.d/pgadmin4.list
```

2. Mettez à jour et installez pgAdmin en mode web :

```
sudo apt update
sudo apt install pgadmin4-web -y
```

3. Configurez pgAdmin :

```
sudo /usr/pgadmin4/bin/setup-web.sh
```

4. Répondez aux questions :
   — Email : votre_email@example.com
   — Mot de passe : choisissez un mot de passe fort
   — Serveur web : apache2
5. Redémarrez Apache :

   ```
   sudo systemctl restart apache2
   ```

6. Autorisez le port HTTP :

   ```
   sudo ufw allow 80/tcp
   ```

7. Accédez à pgAdmin via votre navigateur :

   ```
   http://VOTRE_IP/pgadmin4
   ```

8. Connectez-vous avec l'email et le mot de passe configurés.
9. Ajoutez un serveur PostgreSQL :
   — Clic droit sur "Servers" → "Register" → "Server"
   — General tab : Name = "Local PostgreSQL"
   — Connection tab :
      — Host : localhost
      — Port : 5432
      — Database : postgres
      — Username : postgres
      — Password : Postgres2024 !Secure
      — Save password : oui
   — Cliquez sur "Save"

   Question 3.5 : Quelles mesures de sécurité supplémentaires devrait-on prendre pour pgAdmin en production ?

4 Partie 4 : Manipulation de Données et Audit (30 min)

4.1 Exercice 4.1 : Création de tables et insertion de données

Connectez-vous avec l'utilisateur appweb :

```
psql -h localhost -U appweb -d ecommerce
```

1. Créez une table clients :

   ```
   CREATE TABLE clients (
       id SERIAL PRIMARY KEY,
       nom VARCHAR(100) NOT NULL,
       email VARCHAR(100) UNIQUE NOT NULL,
       ville VARCHAR(50),
       date_inscription TIMESTAMP DEFAULT CURRENT_TIMESTAMP
   );
   ```

2. Créez une table produits :

   ```
   CREATE TABLE produits (
       id SERIAL PRIMARY KEY,
       nom VARCHAR(100) NOT NULL,
       prix NUMERIC(10, 2) NOT NULL CHECK (prix >= 0),
       stock INTEGER DEFAULT 0 CHECK (stock >= 0)
   );
   ```

3. Créez une table commandes :

   ```
   CREATE TABLE commandes (
       id SERIAL PRIMARY KEY,
       client_id INTEGER REFERENCES clients(id) ON DELETE CASCADE,
       date_commande TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
       montant_total NUMERIC(10, 2) NOT NULL
   );
   ```

4. Insérez des données :

```
INSERT INTO clients (nom, email, ville) VALUES
('Alice Martin', 'alice@example.com', 'Paris'),
('Bob Durand', 'bob@example.com', 'Lyon'),
('Charlie Petit', 'charlie@example.com', 'Marseille'),
('Diana Moreau', 'diana@example.com', 'Toulouse');

INSERT INTO produits (nom, prix, stock) VALUES
('Ordinateur portable', 899.99, 15),
('Souris sans fil', 29.99, 50),
('Clavier mécanique', 149.99, 30),
('Écran 27 pouces', 349.99, 20),
('Casque audio', 79.99, 40);

INSERT INTO commandes (client_id, montant_total) VALUES
(1, 899.99),
(2, 179.98),
(1, 429.98),
(3, 29.99);
```

5. Vérifiez les données :

```
SELECT * FROM clients;
SELECT * FROM produits;
SELECT * FROM commandes;
```

6. Effectuez une jointure :

```
SELECT c.nom, c.email, cmd.date_commande, cmd.montant_total
FROM clients c
JOIN commandes cmd ON c.id = cmd.client_id
ORDER BY cmd.date_commande DESC;
```

7. Affichez la structure des tables :

```
\d clients
\d+ produits
```

Question 4.1 : Que signifie SERIAL dans la définition de colonne ?

4.2 Exercice 4.2 : Tests de sécurité et privilèges

1. Avec l'utilisateur appweb, essayez de supprimer la table clients :

```
DROP TABLE clients;
```

Question 4.2 : Cette commande fonctionne-t-elle ? Pourquoi ?

2. Essayez de créer une nouvelle base de données :

```
CREATE DATABASE test_security;
```

Question 4.3 : Que se passe-t-il ?

3. Quittez et reconnectez-vous avec readonly :

```
\q
psql -h localhost -U readonly -d stats
```

4. Créez une table de test (avec postgres d'abord) :

```
# Dans un autre terminal
sudo -u postgres psql stats

CREATE TABLE logs (
    id SERIAL PRIMARY KEY,
    message TEXT,
    niveau VARCHAR(20),
```

```
    date_log TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

INSERT INTO logs (message, niveau) VALUES
('Démarrage du système', 'INFO'),
('Erreur de connexion', 'ERROR'),
('Sauvegarde effectuée', 'INFO');

-- Accordez SELECT à readonly
GRANT SELECT ON logs TO readonly;
GRANT USAGE ON SEQUENCE logs_id_seq TO readonly;
\q
```

5. Retournez au terminal readonly et testez :

```
SELECT * FROM logs;
INSERT INTO logs (message, niveau) VALUES ('Test', 'INFO');
UPDATE logs SET niveau = 'WARNING' WHERE id = 1;
DELETE FROM logs WHERE id = 1;
```

Question 4.4 : Quelles commandes fonctionnent et lesquelles échouent ?

6. Testez l'accès à d'autres bases :

```
\c ecommerce
```

Question 4.5 : Peut-on se connecter à ecommerce avec readonly ?

4.3 Exercice 4.3 : Surveillance et audit

1. Connectez-vous en tant que postgres :

```
sudo -u postgres psql
```

2. Affichez les connexions actives :

```
SELECT pid, usename, application_name, client_addr,
       state, query
FROM pg_stat_activity
WHERE state = 'active';
```

3. Affichez toutes les connexions :

```
SELECT datname, usename, client_addr, state,
       backend_start
FROM pg_stat_activity
ORDER BY backend_start DESC;
```

4. Terminez une connexion spécifique (remplacez PID) :

```
SELECT pg_terminate_backend(PID);
```

ATTENTION : Ne terminez pas votre propre connexion !

5. Consultez les logs PostgreSQL :

```
sudo tail -f /var/log/postgresql/postgresql-15-main.log
```

6. Pour activer plus de logging, éditez postgresql.conf :

```
sudo nano /etc/postgresql/15/main/postgresql.conf
```

7. Cherchez et modifiez ces paramètres :

```
log_connections = on
log_disconnections = on
log_duration = on
log_statement = 'all'  # Attention : verbeux !
```

8. Rechargez la configuration :

```
sudo systemctl reload postgresql
```

9. Effectuez quelques connexions/requêtes et observez les logs.
10. Installez et testez pgAudit (optionnel) :

```
sudo apt install postgresql-15-pgaudit -y
```

```
sudo nano /etc/postgresql/15/main/postgresql.conf
```

```
shared_preload_libraries = 'pgaudit'
pgaudit.log = 'read,write,ddl'
pgaudit.log_catalog = off
```

```
sudo systemctl restart postgresql
sudo -u postgres psql ecommerce
```

```
CREATE EXTENSION pgaudit;
```

Question 4.6 : Pourquoi ne pas laisser log_statement = 'all' en production ?

5 Synthèse et Évaluation

5.1 Questions de réflexion

1. **Architecture PostgreSQL :** Expliquez la différence entre l'utilisateur système postgres et les rôles PostgreSQL. Pourquoi cette séparation existe-t-elle ?
2. **pg_hba.conf vs privilèges :** Quelle est la différence entre le contrôle d'accès dans pg_hba.conf et les privilèges dans PostgreSQL ? Les deux sont-ils nécessaires ?
3. **Rôles comme groupes :** Vous gérez 50 développeurs et 20 analystes. Comment utiliseriez-vous les rôles PostgreSQL pour simplifier l'administration ?
4. **Sécurité multicouche :** Décrivez les 5 couches de sécurité que vous mettriez en place pour protéger un PostgreSQL en production contenant des données sensibles.
5. **Migration MySQL → PostgreSQL :** Quelles sont les principales différences que vous avez observées entre MySQL/MariaDB et PostgreSQL en termes de gestion des utilisateurs et privilèges ?

5.2 Checklist de validation du TP

Vérifiez que vous avez réalisé toutes les étapes :
☐ PostgreSQL installé et démarré
☐ Mot de passe défini pour l'utilisateur postgres
☐ Utilisateurs/rôles créés avec différents privilèges
☐ Bases de données créées avec propriétaires
☐ Privilèges attribués correctement (y compris DEFAULT PRIVILEGES)
☐ Rôles utilisés comme groupes
☐ pg_hba.conf configuré avec scram-sha-256
☐ Configuration réseau modifiée (listen_addresses)
☐ Pare-feu configuré avec règles strictes
☐ pgAdmin installé et configuré
☐ Tables créées et données insérées
☐ Tests de sécurité effectués
☐ Monitoring et logs consultés

5.3 Cas pratique final

Vous devez mettre en place la gestion des utilisateurs pour une application de gestion de projet avec :
— Base de données : `gestion_projets`
— 3 types d'utilisateurs :
  — Chefs de projet : CRUD complet sur toutes les tables
  — Membres d'équipe : lecture de tous, modification de leurs tâches uniquement
  — Observateurs externes : lecture seule sur projets et tâches (pas sur utilisateurs)

— L'application web doit se connecter depuis 192.168.1.100
— Les analystes RH doivent accéder en lecture depuis 192.168.2.0/24
Tâche : Écrivez tous les commandes SQL et configurations nécessaires pour implémenter cette architecture.

5.4 Pour aller plus loin

Si vous avez terminé en avance, explorez ces sujets :

1. **Row Level Security (RLS) :** Implémentez des politiques pour que chaque membre d'équipe ne voit que ses propres tâches.
2. **SSL/TLS :** Configurez PostgreSQL pour accepter uniquement les connexions chiffrées.
3. **Réplication :** Documentez-vous sur la réplication streaming PostgreSQL.
4. **pgBouncer :** Installez un connection pooler pour optimiser les connexions.
5. **Backup/Restore :** Pratiquez pg_dump et pg_restore avec différentes options.
6. **Extensions :** Explorez des extensions utiles (pg_stat_statements, pg_cron, postgis).
7. **Tuning :** Utilisez pgtune pour optimiser postgresql.conf selon votre matériel.

Conclusion

Ce TP vous a permis de découvrir PostgreSQL et ses spécificités :
— Architecture basée sur les rôles (utilisateurs = rôles avec LOGIN)
— Contrôle d'accès multicouche (pg_hba.conf + privilèges)
— Concept puissant de rôles comme groupes
— Granularité fine des privilèges
— Importance de DEFAULT PRIVILEGES
— Méthodes d'authentification flexibles
— Outils d'administration (psql, pgAdmin)
PostgreSQL est réputé pour :
— Sa conformité stricte aux standards SQL
— Sa robustesse et sa fiabilité
— Ses fonctionnalités avancées (JSON, tableaux, types personnalisés)
— Son extensibilité (extensions, procédures, langages)
— Sa communauté active et documentation excellente
Ces compétences sont essentielles pour administrer PostgreSQL en environnement professionnel.

Ressources complémentaires :
— Documentation officielle : https://www.postgresql.org/docs/
— Guide de sécurité : https://www.postgresql.org/docs/current/security.html
— Wiki PostgreSQL : https://wiki.postgresql.org/
— pgAdmin : https://www.pgadmin.org/
— CIS Benchmark PostgreSQL : https://www.cisecurity.org/