ASBD TP1 : Déployer un Postgres sur un serveur Ubuntu 20.04

Auteurs :

- CONTRERAS Baptiste p1809436
- CLEMENT Florent p1601511

Installation manuelle

Ce tutoriel a été effectué sur une distribution Ubuntu 20.4. Utiliser une autre version / distribution demandera surement quelques petites adaptations.
Pour postgres nous avons utilisé la dernière en date à savoir postgres 13.

1) Préparation du serveur

Pour commencer, nous allons mettre à jour les packages de notre machine :

```
sudo apt-get update && sudo apt-get upgrade
```

2) Installation de postgres

Pour l'étape suivante nous allons installer `postgresql`.
Pour ce faire, vous devrez exécuter les commandes suivantes pour obtenir la dernière version stable de postgresql.

```
# Create the file repository configuration:
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -
cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

# Import the repository signing key:
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add -

# Update the package lists:
sudo apt-get update

# Install the latest version of PostgreSQL.
# If you want a specific version, use 'postgresql-12' or similar instead of
'postgresql':
sudo apt-get -y install postgresql
```

Pour plus d'information sur l'installation n'hésitez pas à regarder la documentation officielle. Ici une autre ressource pour Build From Source **(vivement déconseillé pour de la production)**

3) Vérification de l'installation

A partir d'ici nous avons un postgres installé sur notre serveur : il va falloir s'atteler à la configuration !

Vérifions que l'utilisateur postgres existe sur notre système

```
awk -F: '{ print $1}' /etc/passwd | grep postgres
```

Sur cette capture d'écran nous pouvons voir que nous avons bien l'utilisateur `postgres`

```
ubuntu@tiw3-asbd-postgres-25:~$ awk -F: '{ print $1}' /etc/passwd | grep
postgres
postgres
```

Dans le cas où vous n'avez pas d'utilisateur postgres voici la procédure à suivre pour créer l'utilisateur postgres :

```
sudo adduser postgres --system --no-create-home
```

Nous allons ensuite nous connecter pour vérifier que l'installation est fonctionelle :

```
sudo su postgres
psql
```

Le résultat devrait être similaire :

```
postgres@tiw3-asbd-postgres-25:/home/ubuntu$ psql
psql (13.2 (Ubuntu 13.2-1.pgdg20.04+1))
Type "help" for help.

postgres=#
```

Il faudra vérifier que vous êtes bien sur la version attendue.

4) Limitation des ressources allouées

Dans un premier temps nous allons limiter la consommation de mémoire RAM à 25% de notre capacité de RAM . Dans notre cas notre VM avait 0.5 GB de RAM donc un quart = 125MB

Nous allons donc modifier la ligne suivante :

`shared_buffers = 125MB` dans le fichier de configuration `postgresql.conf` (Ce fichier se trouve en général dans `/etc/postgresql/13/main`)

Puis nous redémarrons le service postgres pour appliquer les changements

```
sudo service postgres restart
```

Si jamais vous ne trouvez pas l'emplacement de `postgresql.conf`, vous pouvez tenter de le localiser avec `locate postgresql.conf`

5) Un peu de sécurité

Pour renforcer la sécurité de notre serveur postgres, nous allons utiliser l'algorithme scram-sha-256 pour chiffrer les mots de passe. Toujours dans le fichier `postgresql.conf` on modifie la ligne.

```
password_encryption = scram-sha-256
```

Comme à l'étape précédente nous redémarrons postgres

6) Ajout d'un utilisateur

Une fois notre installation configurée, nous pouvons ajouter un utilisateur et une première base.

Voici une version minimaliste pour créer un utilisateur avec un mote de passe et pouvant se connecter. Il est possible de personnaliser l'utilisateur avec des options avancées listées ici

```
CREATE USER "myuser" WITH LOGIN ENCRYPTED PASSWORD 'toto' VALID UNTIL
'infinity'
```

(toto n'est surement pas un choix judicieux pour un mot de passe)

```
CREATE USER "myuser" WITH LOGIN ENCRYPTED PASSWORD 'toto' VALID UNTIL
'infinity'
CREATE ROLE
```

Une fois l'utilisateur créé vous pouvez faire un `\du` dans postgres pour consulter la liste des utilisateurs.

```
postgres=# \du
                                   List of roles
 Role name |                         Attributes                         |
Member of
-----------+------------------------------------------------------------+-----
------
 myuser    |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

Pour la création de la base de données, on utilise la commande ci-dessous. A noter que l'option `OWNER` permet de spécifier que notre utilisateur créé précédemment est le propriétaire de cette base.
Là encore pour plus d'information sur les options la documention est à disposition !

```
postgres=# CREATE DATABASE mydb WITH OWNER myuser
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding | Collate |  Ctype  |   Access privileges
-----------+----------+----------+---------+---------+-----------------------
 mydb      | myuser   | UTF8     | C.UTF-8 | C.UTF-8 |
 postgres  | postgres | UTF8     | C.UTF-8 | C.UTF-8 |
 template0 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
           |          |          |         |         | postgres=CTc/postgres
 template1 | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres          +
           |          |          |         |         | postgres=CTc/postgres
(4 rows)
```

```
postgres=#
```
Par défaut postgres tente de vous connecter via un Unix socket. Pour qu'il utilise le mot de passe postgres que vous avez créé en même temps que l'utilisateur, il va falloir modifier le fichier `pg_hba.conf` qui se trouve dans le même répertoire que votre ficher `postgresql.conf`.
Vous allez remplacer
```
local   all             all                                     peer
```
par

```
local   all             all                                     scram-sha-256
```
Maintenant nous sommes en mesure de nous connecter à postgres avec votre nouvel utilisateur et son mot de passe.

Ainsi pour accéder à votre utilisateur `myuser` et sa base de données `mydb` nous utiliserons la commande suivante :
```
psql -U myuser -d mydb
```

7) Autoriser les connexions depuis l'extérieur

Nous allons tester la connexion sur notre serveur postgres depuis une autre machine.
Sur notre machine2, `psql -U myuser -h tiw3-asbd-postgres-25.asbdpg.os.univ-lyon1.fr -p 5432 -d mydb` doit échouer avec le message suivant :
```
psql: error: could not connect to server: Connection refused
        Is the server running on host "tiw3-asbd-postgres-25.asbdpg.os.univ-lyon1.fr" (192.168.73.216) and accepting
        TCP/IP connections on port 5432?
```
Apparement aucun service n'écoute sur le port 5432 de notre serveur Ubuntu...
En effet, si nous affichons les ports utilisé sur notre serveur :

```
sudo lsof -i -P -n | grep LISTEN
sshd       39335            root    3u  IPv4 119881      0t0  TCP *:22 (LISTEN)
sshd       39335            root    4u  IPv6 119892      0t0  TCP *:22 (LISTEN)
systemd-r  39584 systemd-resolve   13u  IPv4 120953      0t0  TCP 127.0.0.53:53 (LISTEN)
sshd      662826          ubuntu   10u  IPv6 884078      0t0  TCP [::1]:6010 (LISTEN)
sshd      662826          ubuntu   11u  IPv4 884079      0t0  TCP 127.0.0.1:6010 (LISTEN)
postgres  663676        postgres    5u  IPv4 885251      0t0  TCP 127.0.0.1:5432 (LISTEN)
```
La dernière ligne nous indique que `127.0.0.1:5432 (LISTEN)` postgres écoute que les connexions locales.
Pour corriger ce problème nous avons besoin de deux choses :

- Faire en sorte que postgres écoute les connexions provenant de l'extérieur
- Modifier `pg_hba` pour autoriser la connexion depuis une machine distante

Rendez-vous dans `postgressql.conf` et ajoutez la ligne suivante : `listen_addresses = '0.0.0.0'`
Juste après cela on modifie directement `pg_hba.conf`. Ajoutez

```
host    mydb            myuser          192.168.73.234/24               scram-sha-256
```

Ici `192.168.73.234/24` est l'ip de notre machine distante.
Cette ligne autorise la connexion du user postgres: `myuser` sur la base de données `mydb` en utilisant le connexion par mot de passe postgres, le tout que depuis la machine ayant l'ip ci-dessus. La sécurité avant tout !

Installation automatique (avec ANSIBLE)

1) Installation d'Ansible

Ansible à besoin de python pour fonctionner et il existe deux manières de l'installer :

- Avec le packet manager de votre OS
- avec pip

La meilleure solution reste avec le packet manager car vous avez le dernier build pour votre serveur. C'est donc cette méthode que nous allons utiliser mais au cas où, la méthode alternative est disponible ici

1.1) Installation sur le Control Node

```
sudo apt update
sudo apt install software-properties-common
sudo apt-add-repository --yes --update ppa:ansible/ansible
sudo apt install ansible
```

Dans notre cas ansible est à la version `2.9.6`:

```
ubuntu@tiw3-asbd-postgres-25:~$ ansible --version
ansible 2.9.6
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.8.5 (default, Jan 27 2021, 15:41:15) [GCC 9.3.0]
```

Les documentations et le playbook seront donc adaptés pour cette version

2) Verification de l'installation

Dans la suite de ce document, **CN** fera référence à **Control Node** tandis **MN** à **Managed Nodes**

Pour s'assurer qu'Ansible est opérationnel, nous allons lancer un ping depuis le CN sur nos MN.

Depuis le CN ajoutez les IP ou hosts de vos MN.

```
sudo vim /etc/ansible/hosts
```

Il faudra penser à ajouter ceci au dessus de votre liste de vos IP ou hosts. Cet ajout sera utile pour le playbook.

```
[servers]
```

Ansible communique avec SSH, il faut s'assurer que le CN puisse se connecter en SSH sur les MN avec sa clé privée.

Si ce n'est pas le cas, il faudra ajouter la **clé publique** du CN dans le fichier `./ssh/authorized_keys`
Voici un tutoriel pour créer une clé publique si besoin

Le ping devrait marcher maintenant :

```
ansible all -m ping
tiw3-asbd-postgres-26.asbdpg.os.univ-lyon1.fr | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Notre installation est prête pour la suite !

3) Création du playbook

Avant tout, il vous faudra créer le script `postgresRepo.sh` dans `/home/ubuntu/`
Avec le contenu suivant :

```
# Create the file repository configuration:
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -
cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'

# Import the repository signing key:
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add -

# Update the package lists:
sudo apt-get update

# Install the latest version of PostgreSQL.
# If you want a specific version, use 'postgresql-12' or similar instead of
'postgresql':
sudo apt-get -y install postgresql
```

Avant de créer le playbook nous devons modifier deux paramètres dans la configuration Ansible pour ne pas avoir de problème de droit par la suite.

`sudo vim /etc/ansible/ansible.cfg` et ajouter

```
[ssh_connection]

pipelining=True
allow_world_readable_tmpfiles=True
```

Pour commencer, créez un fichier xxxxxxx.`yml` avec le contenu ci-dessous:

```
---

- hosts: servers
  remote_user: ubuntu
  gather_facts: yes

  vars:
    ansible_ssh_private_key_file: ../.ssh/id_rsa
    dbname: mydb
    dbuser: myuser
    dbpass: toto

  tasks:
  - name: Run apt upgrade
    apt:
      upgrade: "yes"
      update_cache: yes
    become_user: root
    become: true

  - name: Install pip3
    apt:
      name: python3-pip
      state: present
    become: true
```

```
    become_user: root

  - name: Run a script to add official postgres repo
    script: /home/ubuntu/postgresRepo.sh
    become_user: root
    become: true

  - name: Install postgresql
    apt:
      name: postgresql
      state: present
    become: true
    become_user: root

  - name: Make sure psycopg2 is installed
    pip:
      name: psycopg2-binary
      state: present

  - name: Change listening postgres address from localhost to *
    postgresql_set:
        name: listen_addresses
        value: 0.0.0.0
    become: true
    become_user: postgres

  - name: Change postgres conf 1/2
    postgresql_set:
        name: shared_buffers
        value: 125mb
    become: true
    become_user: postgres

  - name: Change postgres conf 2/2
    postgresql_set:
        name: password_encryption
        value: scram-sha-256
    become: true
    become_user: postgres

  - name: restart postgresql
    service:
        name: postgresql
        state: restarted
    become_user: root
    become: true

  - name: Create our user and ensure its priliveges are set correctly
    postgresql_user:
      name: '{{ dbuser }}'
      password: '{{ dbpass }}'
      role_attr_flags: LOGIN
      state: present
      expires: infinity
    become: true
    become_method: sudo
    become_user: postgres

  - name: Create a new database
```

```
    postgresql_db:
      name: '{{ dbname }}'
      encoding: UTF-8
      owner: '{{ dbuser }}'
    become: true
    become_user: postgres

  - name: ensure the user has schema privileges
    postgresql_privs:
      privs: ALL
      type: schema
      objs: public
      role: '{{ dbuser }}'
      db: '{{ dbname }}'
    become: true
    become_user: postgres
  - name: Update pg_hba.conf [local]
    postgresql_pg_hba:
        dest: /etc/postgresql/13/main/pg_hba.conf
        contype: host
        method: scram-sha-256
        contype: local
    become_user: root
    become: true

  - name: Update pg_hba.conf [host]
    postgresql_pg_hba:
        dest: /etc/postgresql/13/main/pg_hba.conf
        contype: host
        method: scram-sha-256
        contype: host
        source: ::1
    become_user: root
    become: true

  - name: Update pg_hba.conf [distant]
    postgresql_pg_hba:
        dest: /etc/postgresql/13/main/pg_hba.conf
        contype: host
        users: '{{ dbuser }}'
        method: scram-sha-256
        contype: host
        databases: '{{ dbname }}'
        source: 192.168.73.216/24
    become_user: root
    become: true
```

Lien de la documentation :

- Les différents keyworks
- apt
- service
- pip
- script
- postgresql_user
- postgresql_db
- postgresql_privs

Pour aller plus loin

```
CREATE TABLE mytable
(
        myid INT GENERATED ALWAYS AS IDENTITY,
        mydate TIMESTAMPTZ  NOT NULL DEFAULT (NOW() AT TIME ZONE 'UTC'),
        PRIMARY KEY(myid)
);
```

Une fois la table créée nous obtenons le résultat suivant:

```
mydb=> \d mytable
                                  Table "public.mytable"
 Column |           Type           | Collation | Nullable |           Default
--------+--------------------------+-----------+----------+--------------------
-----------
 myid   | integer                  |           | not null | generated always
as identity
 mydate | timestamp with time zone |           | not null |
timezone('UTC'::text, now())
Indexes:
    "mytable_pkey" PRIMARY KEY, btree (myid)
```

Nous pouvons ensuite ajouter l'ensemble des données :

```
INSERT INTO mytable(mydate) SELECT mydate
FROM generate_series
        ( '2021-11-27'::timestamp
        , '2021-12-9'::timestamp
        , '1 minute'::interval) as mydate
```