Using Postgres

The minimum supported version of PostgreSQL is determined by the Dependency Deprecation Policy.

Install postgres client libraries

Synapse will require the python postgres client library in order to connect to a postgres database.

- If you are using the matrix.org debian/ubuntu packages, the necessary python library will already be installed, but you will need to ensure the low-level postgres library is installed, which you can do with `apt install libpq5`.
- For other pre-built packages, please consult the documentation from the relevant package.
- If you installed synapse in a virtualenv, you can install the library with:

  ```
  ~/synapse/env/bin/pip install "matrix-synapse[postgres]"
  ```

  (substituting the path to your virtualenv for `~/synapse/env`, if you used a different path). You will require the postgres development files. These are in the `libpq-dev` package on Debian-derived distributions.

Set up database

Assuming your PostgreSQL database user is called `postgres`, first authenticate as the database user with:

```
su - postgres
# Or, if your system uses sudo to get administrative rights
sudo -u postgres bash
```

Then, create a postgres user and a database with:

```
# this will prompt for a password for the new user
createuser --pwprompt synapse_user

createdb --encoding=UTF8 --locale=C --template=template0 --owner=synapse_user
synapse
```

The above will create a user called `synapse_user`, and a database called `synapse`.

Note that the PostgreSQL database *must* have the correct encoding set (as shown above), otherwise it will not be able to store UTF8 strings.

You may need to enable password authentication so `synapse_user` can connect to the database. See https://www.postgresql.org/docs/current/auth-pg-hba-conf.html.

Synapse config

When you are ready to start using PostgreSQL, edit the `database` section in your config file to match the following lines:

```
database:
  name: psycopg2
  args:
    user: <user>
    password: <pass>
    dbname: <db>
    host: <host>
    cp_min: 5
    cp_max: 10
```

All key, values in `args` are passed to the `psycopg2.connect(..)` function, except keys beginning with `cp_`, which are consumed by the twisted adbapi connection pool. See the libpq documentation for a list of options which can be passed.

You should consider tuning the `args.keepalives_*` options if there is any danger of the connection between your homeserver and database dropping, otherwise Synapse may block for an extended period while it waits for a response from the database server. Example values might be:

```
database:
  args:
    # ... as above

    # seconds of inactivity after which TCP should send a keepalive message
to the server
    keepalives_idle: 10

    # the number of seconds after which a TCP keepalive message that is not
    # acknowledged by the server should be retransmitted
    keepalives_interval: 10

    # the number of TCP keepalives that can be lost before the client's
connection
    # to the server is considered dead
    keepalives_count: 3
```

Postgresql major version upgrades

Postgres uses separate directories for database locations between major versions (typically `/var/lib/postgresql/<version>/main`).

Therefore, it is recommended to stop Synapse and other services (MAS, etc) before upgrading Postgres major versions.

It is also strongly recommended to back up your database beforehand to ensure no data loss arising from a failed upgrade.

Backups

Don't forget to back up your database!

Tuning Postgres

The default settings should be fine for most deployments. For larger scale deployments tuning some of the settings is recommended, details of which can be found at https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server.

In particular, we've found tuning the following values helpful for performance:

- `shared_buffers`
- `effective_cache_size`

- `work_mem`
- `maintenance_work_mem`
- `autovacuum_work_mem`

Note that the appropriate values for those fields depend on the amount of free memory the database host has available.

Additionally, admins of large deployments might want to consider using huge pages to help manage memory, especially when using large values of `shared_buffers`. You can read more about that here.

Porting from SQLite

Overview

The script `synapse_port_db` allows porting an existing synapse server backed by SQLite to using PostgreSQL. This is done as a two phase process:

1. Copy the existing SQLite database to a separate location and run the port script against that offline database.
2. Shut down the server. Rerun the port script to port any data that has come in since taking the first snapshot. Restart server against the PostgreSQL database.

The port script is designed to be run repeatedly against newer snapshots of the SQLite database file. This makes it safe to repeat step 1 if there was a delay between taking the previous snapshot and being ready to do step 2.

It is safe to at any time kill the port script and restart it.

However, under no circumstances should the SQLite database be `VACUUM` ed between multiple runs of the script. Doing so can lead to an inconsistent copy of your database into Postgres. To avoid accidental error, the script will check that SQLite's `auto_vacuum` mechanism is disabled, but the script is not able to protect against a manual `VACUUM` operation performed either by the administrator or by any automated task that the administrator may have configured.

Note that the database may take up significantly more (25% - 100% more) space on disk after porting to Postgres.

Using the port script

Firstly, shut down the currently running synapse server and copy its database file

(typically `homeserver.db`) to another location. Once the copy is complete, restart synapse. For instance:

```
synctl stop
cp homeserver.db homeserver.db.snapshot
synctl start
```

Copy the old config file into a new config file:

```
cp homeserver.yaml homeserver-postgres.yaml
```

Edit the database section as described in the section *Synapse config* above and with the SQLite snapshot located at `homeserver.db.snapshot` simply run:

```
synapse_port_db --sqlite-database homeserver.db.snapshot \
    --postgres-config homeserver-postgres.yaml
```

The flag `--curses` displays a coloured curses progress UI. (NOTE: if your terminal is too small the script will error out)

If the script took a long time to complete, or time has otherwise passed since the original snapshot was taken, repeat the previous steps with a newer snapshot.

To complete the conversion shut down the synapse server and run the port script one last time, e.g. if the SQLite database is at `homeserver.db` run:

```
synapse_port_db --sqlite-database homeserver.db \
    --postgres-config homeserver-postgres.yaml
```

Once that has completed, change the synapse config to point at the PostgreSQL database configuration file `homeserver-postgres.yaml`:

```
synctl stop
mv homeserver.yaml homeserver-old-sqlite.yaml
mv homeserver-postgres.yaml homeserver.yaml
synctl start
```

Synapse should now be running against PostgreSQL.

Troubleshooting

Alternative auth methods

If you get an error along the lines of `FATAL: Ident authentication failed for user "synapse_user"`, you may need to use an authentication method other than `ident`:

- If the `synapse_user` user has a password, add the password to the `database:` section of `homeserver.yaml`. Then add the following to `pg_hba.conf`:

  ```
  host    synapse     synapse_user    ::1/128     md5  # or `scram-sha-256` instead of `md5` if you use that
  ```

- If the `synapse_user` user does not have a password, then a password doesn't have to be added to `homeserver.yaml`. But the following does need to be added to `pg_hba.conf`:

  ```
  host    synapse     synapse_user    ::1/128     trust
  ```

Note that line order matters in `pg_hba.conf`, so make sure that if you do add a new line, it is inserted before:

```
host    all         all             ::1/128     ident
```

Fixing incorrect `COLLATE` or `CTYPE`

Synapse will refuse to start when using a database with incorrect values of `COLLATE` and `CTYPE` unless the config flag `allow_unsafe_locale`, found in the `database` section of the config, is set to true. Using different locales can cause issues if the locale library is updated from underneath the database, or if a different version of the locale is used on any replicas.

If you have a database with an unsafe locale, the safest way to fix the issue is to dump the database and recreate it with the correct locale parameter (as shown above). It is also possible to change the parameters on a live database and run a `REINDEX` on the entire database, however extreme care must be taken to avoid database corruption.

Note that the above may fail with an error about duplicate rows if corruption has already occurred, and such duplicate rows will need to be manually removed.