Tutorial: create AQMS postgreSQL database

Section 1. PostgreSQL server installation and configuration

Install the latest stable PostgreSQL database server + PostGIS extensions and psql client

Before you install the database server, create a "postgres" linux user that will run the database server, this user does not need a login. On Ubuntu, the installation process does this automatically.

Follow the instructions on how to add the PostgreSQL code repositories to your local package management system so you can easily get the latest version of PostgreSQL, look at the instructions for binary distributions:

https://www.postgresql.org/download/

If you are concerned about adding the official PostgreSQL code repositories to your yum or apt list of trusted repos, don't be. You'll make your life a lot easier to just use the PostgreSQL repos.

You will need to install the database as `root` or by using `sudo`.

On Ubuntu, you can then install postgreSQL using apt-get, on RedHat style Linux you would use yum.

We will be using the PostGIS extension, which is also available through the PostgreSQL repo.

For example, at the time of this writing, to get the latest PostgreSQL (10) and the latest PostGIS (2.4):

CentOS7:

- Add yum repository for 10:

  ```
  yum install /10/redhat/rhel-7-x86_64/pgdg-centos10-10-2.noarch.rpm
  ```

- Install PostgreSQL:
  `yum install postgresql10 postgresql10-server postgresql10-contrib`
- Install PostGIS:
  `yum install postgis2_10`

http://www.postgresonline.com/journal/archives/362-An-almost-idiots-guide-to-install-PostgreSQL-9.5,-PostGIS-2.2-and-pgRouting-2.1.0-with-Yum.html

- Optionally initialize the database and enable automatic start:

```
/usr/pgsql-10/bin/postgresql-10-setup initdb
systemctl enable postgresql-10
systemctl start postgresql-10
```

Configure database user authentication methods and enable TCP/IP connections for the server.

PostgreSQL typically assumes that the linux user name is the same as the database user name. However, the AQMS model is that we use special database users instead. This means we have to change the default authentication method, except for the superuser postgres. Also, by default TCP/IP connections are not enabled, so that also needs to be changed.

- Edit the pg_hba.conf file (located in the same directory as the postgresql.conf file), change “peer” to “md5” for all lines *except* the line `localhost all postgres peer` (unless you set up a password for user postgres). md5 is a simple password authentication method.

```
local   all             postgres                                peer
local   all             all                                     md5
host    all             all             localhost               md5
host    all             all             127.0.0.1/32            md5
```

- Edit the postgresql.conf file to enable TCP/IP connections from localhost as well as outside. Uncomment the line:

  ```
  #listen_addresses = 'localhost'   # what IP address(es) to listen on;
  ```

  and add the machine’s IP address to the list, e.g.:

  ```
  listen_addresses = 'localhost, 172.25.16.5'    # what IP address(es) to listen on;
  ```

- Restart the database server, e.g.
  - `/usr/lib/postgresql/9.6/bin/pg_ctl -D /var/lib/postgresql/9.6/main restart` (Ubuntu)
  - `/usr/pgsql-9.6/bin/pg_ctl restart -D /var/lib/pgsql/9.6/data/` (CentOS7)

    >> MTH: This does not work for me: /var/lib/pgsql/10/data is owned by user postgres.
    >> Nor can I use sudo since root cannot start/own the postgres process
    >> The only way I could find to do this was:
    sudo systemctl stop postgresql-10
    sudo systemctl start postgresql-10
    Now I can see that it was started with the correct data dir:
    mth@localhost> ps -ef|grep postgr
    postgres 16056 1 1 12:24 ? 00:00:00 /usr/pgsql-10/bin/postmaster -D /var/lib/pgsql/10/data/
    Or maybe I could’ve done

>sudo su postgres
And then run the command

Of course you can use other authentication methods if you want, see the PostgreSQL documentation. Also, if you do that, you will probably need to change some of the scripts and possibly also AQMS configuration files.

Connecting to the database server as postgres

```
sudo su postgres
```

If the localhost log-in authentication for user postgres in the pg_hba.conf file is “peer” (default), which means that the linux username has to match the database username and it does not need a password. You can choose to use a different authentication (see next section), however, here we’ll assume that you are now user postgres and that the authentication method is “peer”.

```
psql
```

If you have several PostgreSQL servers running, you can specify the port that you would like to use, for example (default PostgreSQL port is 5432):

```
psql --port=5433
```

Section 2. Build and install custom AQMS extension

Do this also as `root` user, or use `sudo`.

Obtain libmseed from here: https://github.com/iris-edu/libmseed/releases and install it in a directory, let’s call it MSEEDDIR.

```
cd MSEEDDIR
make shared
```

Checkout the following directory from the CISN repository and let’s put it in a directory name DBpg:

```
svn checkout
svn+ssh://svn@vault.gps.caltech.edu/cisn/DB/branches/uw-dev-branch/DBpg

cd DBpg/create/postgresql-extension
```

Edit the Makefile and make sure the variable MSEEDDIR points to the directory containing the libmseed objects (MSEEDDIR from before). After run make and make install, the `AQMSpg_ext.so` will be installed in `/usr/local/lib`:

```
Make
>> MTH: AQMSpg_ext.c:9:22: fatal error: postgres.h: No such file or directory
Missing all the required postgres headers !!
make install
```

Section 3. Create AQMS database users and schema

Do the rest of this tutorial as linux user `postgres`

Checkout the following directory from the CISN repository:
`svn+ssh://svn@vault.gps.caltech.edu/cisn/DB/branches/uw-dev-branch/DBpg .`

In the following, ***make sure to do things in this particular order!***

Optional: Create Tablespaces

For development or small databases it is not necessary to create tablespaces, skip this part.
For more documentation: https://www.postgresql.org/docs/current/manage-ag-tablespaces.html

Generate customized sql scripts

Note: this procedure also creates a password-file so that you can run subsequent sql-scripts without having to input a password. However, after you are done with the database work, you may want to remove that file (in ~postgres, which on Ubuntu is in /var/lib/postgresql) depending on how secure your host machine is.

1. `cd DBpg/create`
2. Edit the file "`pg.env`" and replace the default values for the variables where needed, make sure to save as plain text. This is a very important step, read the comments inside the file carefully!
3. To put values of variables defined in pg.env into the appropriate sql-scripts, run:
   `./generate_sql_scripts.sh`

Create database, activate PostGIS extensions, create roles, users, schema

```
In DBpg/create:
   1. sudo su postgres
   2. ./run_as_postgres.sh
```

This script will prompt several times whether you want to continue, skip a step, or exit. You can safely do a step twice, but you'll get some error messages (for example: ERROR: database "rtdb2" already exists).

List of actions this script does:

```
echo "--> Creating AQMS database $DBNAME..."
echo "--> Activating PostGIS extensions in database $DBNAME as user postgres..."
echo "--> Creating database roles as user postgres..."
echo "--> Creating database users and schemas, and setting user search_path in $DBNAME as user
postgres..."
```

NOTE, at least on ubuntu, if you get this error message:
ERROR: could not load library "/usr/lib/postgresql/9.5/lib/postgis-2.2.so": /usr/lib/liblwgeom-2.2.so.5: undefined symbol: GEOSDelaunayTriangulation

you probably have an old version of libgeos installed somewhere, you need version 3.4+ , which was installed as a dependency. Remove the offending old version and re-run the script to re-try the “Activating PostGIS extensions” and subsequent steps.

Create sequences, tables, views, stored functions

After you have created the database and activated the PostGIS extensions for it, you are ready to create sequences, tables, etc. etc. by running this script.

```
In DBpg/create:
./run_sql_scripts.sh
```

This script will prompt several times whether you want to continue, skip a step, or exit. You can safely do a step twice, but you’ll get some error messages (for example: ERROR: table "arrival" already exists).

List of actions this script does:

```
echo "--> Creating sequences in database $DBNAME as user trinetdb..."
echo "--> Granting privileges on sequences in database $DBNAME as user trinetdb..."
echo "--> Installing waveform_schema tables in $DBNAME as user trinetdb..."
echo "--> Installing parametric_schema tables in $DBNAME as user trinetdb..."
echo "--> Installing instrument_response_schema tables in $DBNAME as user trinetdb..."
echo "--> Installing hardware_schema tables in $DBNAME as user trinetdb..."
echo "--> Installing application_schema tables in $DBNAME as user trinetdb..."
echo "--> Granting privileges on all tables in schema trinetdb in database $DBNAME as user
trinetdb..."
echo "--> Generating pre-defined views needed by Jiggle in database $DBNAME as user trinetdb..."
echo "--> Loading stored procedures (functions) into database $DBNAME as user code..."
```

It may also prompt you for a password several times (if there is not a correct .pgpass in ${HOME}) (the passwords were provided in pg.env) so if you want to pipe the output to a file, make sure to open another window in which you can tail the output file to see if the script is waiting for input from you, e.g.:

```
./run_sql_scripts.sh > run.out
```
In another terminal window:
```
tail -f run.out
```

Alternatively, you can do everything step-by-step, by following the instructions in Appendix A although it is *not recommended* because it is very easy to mess up a step (in particular, to forget specifying the database name or the wrong database user).

Section 4. Load required meta-data

Some tables need to be loaded with information to enable the automatic processing of data. This section aims to provide a comprehensive overview of all the meta-data needed, plus point to example loader scripts in DPpg/gazetteer_data

Overview

Table name	**How is it used?**	**Loader script?**
epochtimebase	It determines whether leapseconds are used, set to 'T' during creation of the table (i.e. set to TRUE).	create_EPOCHTIMEBASE.sql
rt_role	To determine whether the database is in "primary" role.	TBD
leap_seconds	Used by the TRUETIME functions to translate between nominal (w/o leapseconds) and truetime (including leapseconds). Also used by Jiggle.	Init_leap_seconds.sql
magprefpriority	This can optionally be left empty, used by all relevant programs to determine which magnitude type should be set to "preferred" magnitude. See: http://vault.gps.caltech.edu/trac/cisn/wiki/MagPrefPriority	example_magprefpriority.sql
stacorrections	Me and Ml magnitude station corrections, used by trimag and Jiggle but not by localmag (earthworm), See for example: http://vault.gps.caltech.edu/trac/cisn/wiki/RT_trimag_man	magnitude_corrections.sql
gazetteer_region	Define region polygons, used by ec and Jiggle	pg_network_regions.sql
assoc_region_group	Using this table allows you to assign different network codes to different events, used by ec and Jiggle. See: http://vault.gps.caltech.edu/trac/cisn/wiki/TierII	example_assoc_region_group.sql
station_data channel_data simple_response	Meta-data for channels processed by this system. Used by the real-time programs tc, ec, trimag, rad2, and ampgen. Also used by Jiggle and these tables serve as the source for several views used by Jiggle.	https://github.com/pnsn/aqms-ir loadStationXML script or: http://github.com/pnsn/fdsnsws-station2aqms (which uses the aqms-ir package but I (RH) am not actively adding new options to this script. I recommend using getStationXML and loadStationXML from the above.
channelmap_ampparms		https://github.com/pnsn/aqms-ir cliplevels in utils is a messy way of getting what PNSN needs.
channelmap_codaparms		Also filled by https://github.com/pnsn/aqms-ir I

		believe

program	program-lookup-codes and channel lists for each program, program and config_channel are mostly used by the RT systems and only contain currently active channels.
	
	Directly used by:
	`rad2 : ProgramName RAD-UW`
	`Ampgen : ProgramName AmpGen`
	`Trimag : ProgramName RAD-UW`
	Used to generate earthworm configuration files:
	`pick_ew_uw.sta`
	`uw_sta.hinv`
	`pnsn_trig.sta`	

config_channel		
applications	program-lookup-codes and channel lists for each program, applications and appchannels are mostly used by PP systems and can contain the history of channels at a site.	
appchannels		

- `cd DBpg/gazetteer_data`
- `psql -U trinetdb -d` *`DBNAME`* `-W < pg_load_information.sql`

Channel meta-data

This describes how to load the minimally required meta-data into the database tables Station_Data, Channel_Data, and Simple_Response using a FDSN StationXML web service:

See: https://github.com/pnsn/fdsnws-station2aqms

Based on: https://github.com/pnsn/aqms-ir which includes a loadStationXML script that PNSN actually uses. The above fdsnws-station2aqms script may be a bit neglected.

Appendices

Appendix A: List of individual steps

These are the steps performed by scripts DBpg/create/run_as_postgres.sh and DBpg/create/run_sql_scripts.sh. All instructions assume that your working directory is DBpg/create !! You will have to change to the correct paths before running the sql-scripts. It is assumed that the DBpg checkout directory is also called DBpg, if not, replace DBpg with whatever it is you used. It is also important to use the postgres linux-user when needed, and to specify the correct database and database user where needed.

NOTE: if you are not using the default PG port (5432) make sure to provide the --port= or -p option to the psql command!

Customize

1. in DBpg/create as the user has local write privileges
2. edit pg.env
3. ./generate_sql_scripts.sh

Create database

Replace DBNAME with database name specified in pg.env

- in DBpg/create
- sudo su postgres
- psql < db/pg_create_*DBNAME*.sql

Activate PostGIS extension

- in DBpg/create as postgres linux user
- psql < db/pg_activate_extensions.sql

Create roles

1. in DBpg/create as postgres linux user
2. psql < users/pg_create_roles.sql

To list the newly created roles, connect to the database with psql and type `\du`

Create roles with login (users), schemas, grant privs, set search_paths

1. in DBpg/create as postgres linux user
2. psql -d *DBNAME* < users/pg_create_users.sql where *DBNAME* is the name of the database you created (otherwise trinetdb schema is created in postgres database rather than your target database because script does not only create users but also creates trinetdb and code schemas, grants privileges and sets search_path of each role).

To list the newly created roles, connect to the database with psql and type `\du`

Create sequences and grant privileges

VERY IMPORTANT: Sequences are used by AQMS software to derive primary keys from, which are required to be unique, so if you are going to populate your PostgreSQL database with data from a previous database, you need to make sure that the start values are set to the *nextval* of your old database's sequence. Also, because of the AQMS replication model, primary keys (sequences) generated from different databases should not clash with each other. This is achieved by staggering the start value for each database, for example:

```
# sequence starts
# rtdb 1
if [ "$dbname" = "rtdb" ]; then
  seq_start_value=1
  evid_start_value=60000001
fi

# rtdb2
if [ "$dbname" = "rtdb2" ]; then
  seq_start_value=2
  evid_start_value=60000002
fi

# archdb
if [ "$dbname" = "archdb" ]; then
  seq_start_value=3
  evid_start_value=70000003
fi

# archdb2
if [ "$dbname" = "archdb2" ]; then
  seq_start_value=4
  evid_start_value=80000004
fi
```

You have to create the sequences as user trinetdb for them to be in the trinetdb schema and owned by trinetdb, you also have to grant select privileges to trinetdb_write and code (replace *DBNAME* by correct database name):

1. in DBpg/create
2. psql -U trinetdb -W -d *DBNAME* < sequences/pg_create_dbname_sequences.sql
3. psql -U trinetdb -W -d *DBNAME* < sequences/pg_grant_all_sequences.sql

Create database tables and grant privileges

```
cd DBpg/create/tables/waveform_schema
psql -U trinetdb -W -d DBNAME  < pg_install_waveform_tables.sql

cd DBpg/create/tables/parametric_schema
psql -U trinetdb -W -d DBNAME  < pg_install_parametric_tables.sql

cd DBpg/create/tables/instrument_response_schema
psql -U trinetdb -W -d DBNAME  < pg_install_instrument_response_tables.sql
```

```
cd DBpg/create/tables/hardware_schema
psql -U trinetdb -W -d DBNAME  < pg_install_hardware_tables.sql

cd DBpg/create/tables/application_schema
psql -U trinetdb -W -d DBNAME  < pg_install_application_tables.sql

cd DBpg/create/tables
psql -U trinetdb -W -d DBNAME  < pg_grant_all_tables.sql
```

Create database views

```
cd DBpg/views
psql -U trinetdb -W -d DBNAME  < pg_generate_views.sql
psql -U trinetdb -W -d DBNAME  < pg_grant_views.sql
```

Load stored procedures

```
cd DBpg/storedprocedures
psql -U code -d DBNAME -W  < install_as_user_code.sql
psql -U trinetdb -d DBNAME -W  < install_as_user_trinetdb.sql

as postgres:
psql -d DBNAME < install_as_user_postgres.sql
```

Appendix B: psql cheat sheet

Connecting to the database server as postgres (assuming peer authentication)

```
sudo su postgres
Psql
# list db names while in psql
psql>\l
# connect to database DBNAME
psql>\c DBNAME
```

Connecting to specific database as a specific database user (role) with a password

psql -d dbname -U rolename -W

If you have a .pgpass file in $HOME with the correct permissions and the PGPASSFILE variable set, you can use lower-case w to avoid being prompted for the password:

```
psql -d dbname -U rolename -w
```