GAVO DaCHS installation and configuration

Author: Markus Demleitner
Email: gavo@ari.uni-heidelberg.de
Date: 2016-03-16

Contents

These installation instructions cover the installation of the complete data center suite. Installing libraries or, say, the tapsh, is much less involved. See the respective pages at the GAVO DC's software distribution pages for details on those.

Debian systems

The preferred way to run DaCHS is on Debian stable or compatible systems. To install, add our APT repository to your */etc/apt/sources.list* and install the *gavodachs-server* package, e.g. by running:

```
sudo apt-get install gavodachs-server
```

With that, you're ready to proceed to the tutorial and the Operator's Guide

Installation without Package Management

Dependencies

Unfortunately, DaCHS has quite a few dependencies; here's the list of dependencies of our Debian package as of version 0.8.4; this should give you some clue as to what might be necessary on other systems:

```
adduser, members, postgresql-8.4-pgsphere |
postgresql-9.0-pgsphere | postgresql-9.1-pgsphere, postgresql-8.4-q3c |
postgresql-9.0-q3c | postgresql-9.1-q3c, python, python-imaging,
python-nevow, python-numpy, python-pkg-resources, python-psycopg2,
python-pyfits, python-soappy, python-twisted,
python-zsi, python-setuptools, python-all, python-lxml,
python-docutils, python-pywcs, python-matplotlib
```

If you want to use boosters, you will additionally need:

```
build-essential libcfitsio3-dev
```

Of course, you'll need postgres itself on top of that. We currently require postgres 9.0 or newer. If you actually need support for older Postgres releases, let us know – it's not hard to restore.

PgSphere

PgSphere is a postgres extension for spherical geometry. It is needed for support of the geometric types in DaCHS' ADQL implementation and in the preferred SIAP backend, so you should definitely install it. Obtain the source as https://github.com/mnullmei/pgsphere/archive/fixes-1-1-1.tar.gz (for the time being), install the server development packages for postgres (such as postgresql-server-dev-9.x or postgresql-devel), and in the source directory run:

```
USE_PGXS=1 make
sudo USE_PGXS=1 make install
```

Q3C

DaCHS uses the Q3C library by Sergey Koposov and Oleg Bartunov, http://www.sai.msu.su/~megera/oddmuse/index.cgi/SkyPixelization for positional indexes. DaCHS uses it for positional indexes (the scs#q3cindex mixin) and in the interpretation of ADQL. It is therefore highly recommended to install it.

To do that, get the source directly from https://github.com/segasai/q3c/releases/, install the server development packages for postgres (such as postgresql-server-dev-9.x or postgresql-devel), and in the source directory run:

```
make
sudo make install
```

Installing without a package manager

Getting the source

If you cannot use the Debian package (or do not want to), you can grab a gavodachs package from our distribution page. Choose whatever *gavodachs-latest.tar.gz* points to.

If you want to follow the bleeding edge closely – DaCHS is being actively developed – check out whatever is in the subversion repository right now. For a read-only copy, say:

```
svn co http://svn.ari.uni-heidelberg.de/svn/gavo/python/trunk/ dachs
```

After that, the current source code is in the `dachs` subdirectory. This is development code, so *please* do not hesitate to contact us if something weird is going on with it. We mean it; even trivial reports help us to gauge where our software behaves contrary to expectations. Plus, we don't have oodles of users, so chances are you won't get on our nerves. For contact options see http://docs.g-vo.org/DaCHS/#support.

Installing from source

The DaCHS installer is based on setuptools; we do not use setuptools' dependency management, though, since in practice it seems more trouble than it's worth, which means you need to manually install Dependencies.

To install the software, in the `dachs` directory you checked out above, say:

```
sudo python setup.py develop
```

(there are various options to get the stuff installed when you prefer not to install as root; refer to the setuptools documentation if necessary). The checkout itself needs to be readable by whoever later runs the server in this mode. You can also use `install` instead of `develop`; in that case, you will have to rerun setup.py everytime you update the source.

Setup

All this is taken care of by the Debian package, so don't do any of this if you installed from .deb.

Introduction

GAVO DaCHS is quite sensitive to a correct setup as regards permissions. Experience has shown that user setup is the number one reason for installation problems. So, up front, here's what the steps given below should create:

- A group that will own certain directories that must be writable by the server (by default gavo).
- A user that the server will run as (by default gavo).
- A unix account for you that should not be root (in particular not if you're using setup.py develop on an SVN checkout). This should be in the gavo group (for when you're running `gavo serve debug`) and will usually own resource directories and the like.

On the database side, the following must be ascertained:

- There's a postgres database cluster in the C locale, with a database already created (named, by default gavo).
- "you" (i.e., your unix id) have admin privileges on this (at least for installation) using ident authentication
- for connections from the local host, the three roles the server use can access the database using md5 authentication.

Account Management

You should first create a user that the DaCHS server runs as later, and a group for running DC-related processes in:

```
sudo adduser --system gavo
sudo addgroup --system gavo
sudo adduser gavo gavo
```

(or similar, depending on your environment). This user should not be able to log in, but it should have a home directory. Everyone that may issue a `gavo serve debug` must be in the group created (this is because the log directory will be writable by this group); in particular, you should add yourself:

```
sudo adduser `id -nu` gavo
```

You may want to create another account for "maintenance", or just use your normal account; if more than one person will feed the data center, you'll need more elaborate schemes.

To update the system's idea of your group membership, say `newgrp gavo` or log in and out now.

All users that are to ingest data into the database using DaCHS must be part of this gavo group.

Database setup

The most complicated step in setting up DaCHS is actually setting up the database. We currently only support postgres.

While it is conceivable to use DaCHS together with an existing postgres database, we do not recommend trying this the first time. Experiment with a database dedicated to DaCHS first, then consider whether it's worth interfacing to your existing database or whether a copy of that data is more convenient.

Cluster Creation

You first need a database to play with, preferably in a suitable cluster (you could skip this, but the all bets are off as to whether you'll be able to store non-ASCII characters in strings).

It is recommended to create a dedicated cluster first even if you want to connect DaCHS to a pre-existing database later to get a feel for how it works. See Connecting to a remote database for information on what setup is necessary in this case.

Database cluster generator is very system-dependent, and ideally a database admin would assist you.

On Debian systems dedicated for GAVO DaCHS, you can try the following:

(1) Find out the version of the server you will be running (e.g., using `dpkg -l`; in Debian, more than one version may be installed in parallel. It's probably a good idea to use the most recent one. Set your desired version for subsequent use:

```
export PGVERSION=9.1
```

(2) Drop the Debian default cluster (this will delete everything in there -- for a fresh install, that doesn't matter, but don't do this if other people use the database). If you don't do this, your database will listen do a different port, and you will have to adapt the default profiles:

```
sudo pg_dropcluster --stop $PGVERSION main
```

(3) Create the new cluster used by DaCHS:

```
sudo pg_createcluster -d /<path-to-where-your-db-should-reside> \
  --locale=C -e UNICODE\
  --lc-collate=C --lc-ctype=C $PGVERSION main
```

The locale should currently be C, because only the C locale will allow you databases with all kinds of encodings. The database stores descriptions and similar entities, and you may encounter funny characters in there. It would be a shame if you couldn't store them (plus, you would get odd error messages for those).

(4) Start the server:

```
sudo /etc/init.d/postgresql start
```

(5) Create the database itself:

```
sudo -u postgres createdb --encoding=UTF-8 gavo
```

On Debian, the configuration files for this cluster are at `/etc/postgresql/$PGVERSION/pgdata/`.

Initial Account Setup

At least during setup, you also need superuser privileges on the database. For `gavo init` below to work, your normal account must have such privileges. On Debian systems, you can simply say:

```
sudo -u postgres createuser -s `id -nu`
```

You can drop those privileges later if they make you nervous, but for gavo init you need to be DB superuser. Also note that DaCHS assumes your server is trusted, and if people have managed to take over an account in the gavo group, they can do with your database whatever they please anyway. In particular (don't complain we didn't tell you), DaCHS currently encrypts *no* passwords; for the DB passwords, sensible encryption would mean the software requires some passphrase during startup, which we don't want. For user passwords (for protecting web resources), it would make no sense since with HTTP basic authentication as employed by DaCHS, they travel through the net unencrypted anyway (which is sometimes called "mild security").

Connecting to a Remote Database

If you already have a database and are fed up with playing around in the toy database we advised you to start with at the top of Cluster Creation, you still need to prepare the database server. You may need to edit pg_hba.conf on the remote side as discussed in adapting pg_hba.conf to allow password authenticated access from the host that runs DaCHS. And, during initialization, you will need access to a superuser profile, in the following, admin:

```
gavo init -d "host=foo.bar user=admin password=secret"
```

Adapting pg_hba.conf

The default DaCHS configuration speaks to the server via TCP/IP. Unless you give an option during gavo init below, it will try to connect to the server on 127.0.0.1. This will work fairly well, but depending on usage details, you may gain some (or even quite a lot) performance if you let DaCHS talk to postgres

via unix domain sockets (that, of course, only works if DaCHS and postgres are on the same server). To do that, you must fix your `pg_hba.conf`. This is tricky, and we recommend you make your system run first and then try what's described here if you think you must.

On Debian systems, the `pg_hba.conf` is in `/etc/postgresql/<version>/<cluster>/`, e.g. `/etc/postgresql/9.2/pgdata/` if you created a new cluster by the abover recipe with postgres 9.2, or `/etc/postgresql/8.4/main/` in a default Debian cluster with postgres 8.4. Other systems may have it in whacky places.

Note that running a database server always is a security liability. You should make sure you understand what the pg_hba.conf (in postgres' configuration directory) says.

To let DaCHS access postgres via unix domain sockets, you must have a line like:

```
local   gavo        gavo,gavoadmin,untrusted          md5
```

in pg_hba.conf, probably right below the line allowing the postgres user complete access (the order of lines in pg_hba.conf is significant); it allows password authentication for the three users above from the local machine. To make the server use the new local connection, comment out the line for `host` in your `rootDir/etc/dsn` file (with a hash).

You must restart the postgres server after you changed pg_hba.conf to make the server notice your changes.

Configuration File

Next, you need to decide on a "root" directory for DaCHS. Below it, there are data descriptions, cache files, logs, etc. (these locations can be changed later, but for a simple setup we recommend keeping everything together). By default, this is `/var/gavo`.

DaCHS is configured in an INI-style configuration file in `/etc/gavo.rc` (overridable using the envirvonment variable `GAVOSETTINGS`). In addition, users, in particular the gavo user, can have ~/.gavorc files, the contents of which override settings in /etc/gavo.rc.

Configuration Settings gives a walkthrough through the most important settings; for now, you must set the DaCHS root dir if you are not happy with `/var/gavo`:

```
[general]
rootDir: /data/gavo
```

as /etc/gavo.rc.

You can now let DaCHS create its file system hierarchy:

```
gavo init
```

For this to work, `rootDir` must exist and be writable by you, or you must have sufficient privileges to create it. Do *not* run `gavo init` as root, since the files and directories it creates will be owned by whoever ran the program. In the typical situation in which you may not write to `rootDir`'s parent, do something like:

```
sudo mkdir -p /data/gavo
sudo chown `id -nu`:gavo /data/gavo
```

`gavo init` may spit out a warning or two on the first run. On repeated runs no output at all should appear.

If your database server is not on the same machine as your web server (which is not recommended for a test setup), you have to pass a complete DSN that lets DaCHS connect as a superuser to `gavo init`. A DSN ("Data Source Name") is a sequence of key-values pairs as used by ODBC or Postgres itself (with keys discussed in Database Connection Control Functions in the postgres documentation). You would say something like:

```
gavo init --dsn "host=myhost.xy port=5546 user=super password=secret dbname=wisdom"
```

– make sure you give at least dbname and whatever role DaCHS ends up using has superuser privileges during setup (that role is not used during normal DaCHS operation any more).

You can later run gavo init again. It will not clobber anything you did in the meantime (well, if it does, it's a bug and you should fiercely complain). In particular, this is the most convenient way to create directories if you changed locations in `gavo.rc`.

Binaries

If you want to provide previews for images, you must compile some binaries. these are in the `src` subdirectory of the source tree. Just enter all subdirectories there in turn and say `make install` in each. Run these as yourself (i.e., as data-center administrator), not as root. [If you're looking for a nice project to contribute, replacing those with python equivalents would be welcome and doesn't take much knowledge of DaCHS' internals. products.ScaledFITSProduct shows how this could work for the FITS case, and jpegpreview should probably be done by PIL in-memory]

Specifying meta information fallbacks

In the file `$GAVO_ROOT/etc/defaultmeta.txt` you should give some information filled in when the resources do not give this kind of metadata themselves. Don't sweat it for now, but you must fix it before you run your own registry.

The init script

Though you can operate the server manually through `gavo serve (start|stop|reload)`, you will probably want to install an init script to `/etc/init.d` (or your system's equivalent place). More information on this is in Starting and Stopping the Server in the operator's guide

This concludes the installation instructions for the normal case. Only read on if you're curious and/or courageous.

Manual Database Preparation

Normally, the following steps are done by `gavo init`. So, on a normal install you can stop reading here.

However, if you want to play tricks (e.g., remote database server), the following instructions should help.

Creating database users

The data center software accesses the database in various functions. These are mapped to profiles which correspond to access information (basically, the DSN, user, and password). There are three of them:

- feed -- the "admin" profile, used for feeding tables into the normal database, for user management, credentials checking and the like.
- trustedquery -- this profile is used for queries generated by the DC software (though usually on behalf of a user). The corresponding DB role can access all "normal" tables, privilege management is supposed to happen throught the web interface.
- untrustedquery -- the profile used for user-contributed SQL. Only tables expressly opened up are accessible to it.

You can adapt those names as necessary in the corresponding profiles. See the section on profiles in the Operator's Guide for details.

The following procedure sets up users and databases as expected by the default profiles (if you made yourself a superuser account as described above you do not need the `sudo -u postgres` in these commands):

```
# create the database that'll hold your data
sudo -u postgres createdb --encoding=UTF-8 gavo
# create the user that feeds the db...
sudo -u postgres createuser -P -ADsr gavoadmin
# and a user that usually has no write privileges
sudo -u postgres createuser -P -ADSR gavo
# and a user for ADQL queries (i.e., untrusted queries from the net)
sudo -u postgres createuser -P -ADSR untrusted
```

Enter the passwords you assign here into the `feed`, `trustedquery`, and `untrustedquery` profiles, respectively. These profiles are found in `rootDir/etc`.

Finally, you need to let the various roles you just created access the database; you do this using the command line interface to postgres:

```
sudo -u postgres psql gavo \
  -c "GRANT ALL ON DATABASE gavo TO gavoadmin"
```

For the individual tables, rights to gavo and untrusted are granted by `gavo imp`, so you do not need to specify any rights for them.

Owner-only DB setup

There is some setup that the database owner or at least a superuser must do. Right now, that is allowing stored procedures in Postgres' own procedural language:

```
psql gavo <<EOF
CREATE LANGUAGE plpgsql
EOF
```

Reading the extensions' SQL files

Both pgsphere and q3c have files that define SQL functions and such. You'll have to manually read them into your new database. You can find these SQL files in the source directories of the packages, or in your server's contrib directory. On Debian systems, these contribution directories are in `/usr/share/postgresql/<VERSION>/contrib`.

So, on postgres 8.4 you could say:

```
SRCDIR=/usr/share/postgresql/8.4/contrib
psql gavo < $SRCDIR/q3c.sql
psql gavo < $SRCDIR/pg_sphere.sql
```

Importing basic resources

There are some built-in tables in DaCHS, related to metadata storage, certain protocols, and the like. You must import them before the DC software can be used. This also is a nice test that at least some things work.

So, in this sequence, run:

```
gavo imp --system //dc_tables
gavo imp --system //services
gavo imp --system //users
gavo imp --system //adql
gavo imp --system //tap
gavo imp --system //products
gavo imp --system //obscore
```

Output of the type `Columns affected: 0` is ok for these commands.

The double slash in the identifiers above means "use system resources". All these really refer to resource descriptors (RD) in the ___system___ resource directory; at this point, they are the RDs shipped with DaCHS.

If you get error messages, add a `--hints` after the gavo command, like this:

```
gavo --hints imp --system //dc_tables
```

This will (for the `gavo` command in general) give additional error info where available.

You should now be able to run the examples in the tutorial.