Ubuntu Developer Setup

Base OS: Ubuntu 20.04 LTS

This guide will setup an ODC core development environment and includes:

- Mambaforge using conda environments to isolate the odc development environment
- installation of required software and useful developer manuals for those libraries
- Postgres database installation with a local user configuration
- Integration tests to confirm both successful development setup and for ongoing testing
- Build configuration for local ODC documentation

Required software

GDAL, HDF5, and netCDF4:

```
sudo apt-get install libgdal-dev libhdf5-serial-dev libnetcdf-dev
```

Install the latest Postgres version available for your Ubuntu distribution, eg:

```
sudo apt-get install postgresql-14

# Optionally, Postgis too (required for the postgis/experimental index driver)
sudo apt-get install postgresql-14-postgis-3
```

Ubuntu's official repositories usually ship older versions of Postgres. You can alternatively get the most recent version from the official PostgreSQL repository.

Optional packages (useful utilities, docs):

```
sudo apt-get install libhdf5-doc netcdf-doc libgdal-doc
sudo apt-get install hdf5-tools netcdf-bin gdal-bin pgadmin3
```

Python and packages

Python 3.8+ is required.

Conda environment setup

Conda environments are recommended for use in isolating your ODC development environment from your system installation and other Python environments.

We recommend you use Mambaforge to set up your conda virtual environment, as all the required packages are obtained from the conda-forge channel. Download and install it from here.

Download the latest version of the Open Data Cube from the repository:

```
git clone https://github.com/opendatacube/datacube-core
cd datacube-core
```

Create a conda environment named `cubeenv`:

```
mamba env create -f conda-environment.yml
```

Activate the `cubeenv` conda environment:

```
conda activate cubeenv
```

Find out more about conda environments here.

v: latest

Postgres testing database configuration

This configuration supports local development using your login name.

If this is a new installation of Postgres on your system it is probably wise to set the postgres user password. As the local "postgres" Linux user, we are allowed to connect and manipulate the server using the psql command.

In a terminal, type:

```
sudo -u postgres psql postgres
```

Set a password for the "postgres" database role using the command:

```
\password postgres
```

and set the password when prompted. The password text will be hidden from the console for security purposes.

Type **Control+D** or **\q** to exit the posgreSQL prompt.

By default in Ubuntu, Postgresql is configured to use `ident sameuser` authentication for any connections from the same machine which is useful for development. Check out the excellent Postgresql documentation for more information, but essentially this means that if your Ubuntu username is `foo` and you add `foo` as a Postgresql user then you can connect to a database without requiring a password for many functions.

Since the only user who can connect to a fresh install is the postgres user, here is how to create yourself a database account (which is in this case also a database superuser) with the same name as your login name and then create a password for the user:

```
sudo -u postgres createuser --superuser $USER
sudo -u postgres psql

postgres=# \password <foo>
```

Now we can create databases for integration testing. You will need 2 databases - one for the Postgres driver and one for the PostGIS driver. By default, these databases are called `pgintegration` and `pgisintegration`, but you can name them however you want:

```
postgres=# create database pgintegration;
postgres=# create database pgisintegration;
```

Or, directly from the bash terminal:

```
createdb pgintegration
createdb pgisintegration
```

Connecting to your own database to try out some SQL should now be as easy as:

```
psql -d pgintegration
```

If createdb or psql cannot connect to server, check which postgresql installation is being run:

```
which psql
```

If it is running the mambaforge installation, you may need to run the global installation:

```
/usr/bin/psql -d pgintegration
```

You can now specify the database user and password for ODC integration testing. To do this:

```
cp integration_tests/integration.conf ~/.datacube_integration.conf
```

Then edit the `~/.datacube_integration.conf` with a text editor and add the following lines, re v: latest `<foo>` with your username and `<foobar>` with the database user password you set above (not the postgres one, your `<foo>` one):

```
[datacube]
db_hostname: /var/run/postgresql
db_database: pgintegration
index_driver: default
db_username: <foo>
db_password: <foobar>

[experimental]
db_hostname: /var/run/postgresql
db_database: pgisintegration
index_driver: postgis
db_username: <foo>
db_password: <foobar>
```

Verify it all works

Install additional test dependencies:

```
cd datacube-core
pip install --upgrade -e '.[test]'
```

Run the integration tests:

```
./check-code.sh integration_tests
```

Note: if moto-based AWS-mock tests fail, you may need to unset all AWS environment variables.

Build the documentation:

```
pip install --upgrade -e '.[doc]'
cd docs
pip install -r requirements.txt
sudo apt install make
sudo apt install pandoc
make html
```

Then open `_build/html/index.html` in your browser to view the Documentation.

ODC License | Change Log

v: latest