Switch2OSM

Take back control of your maps

Manually building a tile server (18.04 LTS)

This page describes how to install, setup and configure all the necessary software to operate your own tile server. The step-by-step instructions are written for Ubuntu Linux 18.04 LTS (Bionic Beaver).

Software installation

The OSM tile server stack is a collection of programs and libraries that work together to create a tile server. As so often with OpenStreetMap, there are many ways to achieve this goal and nearly all of the components have alternatives that have various specific advantages and disadvantages. This tutorial describes the most standard version that is similar to that used on the main OpenStreetMap.org tile servers.

It consists of 5 main components: mod_tile, renderd, mapnik, osm2pgsql and a postgresql/postgis database. Mod_tile is an apache module that serves cached tiles and decides which tiles need re-rendering - either because they are not yet cached or because they are outdated. Renderd provides a priority queueing system for different sorts of requests to manage and smooth out the load from rendering requests. Mapnik is the software library that does the actual rendering and is used by renderd.

Note that these instructions are have been written and tested against a newly-installed Ubuntu 18.04 server. If you have got other versions of some software already installed (perhaps you upgraded from an earlier Ubuntu version, or you set up some PPAs to load from) then you may need to make some adjustments.

In order to build these components, a variety of dependencies need to be installed first:

```
sudo apt install libboost-all-dev git-core tar unzip wget bzip2 build-essential aut
```

Say yes to install. This will take a while, so go and have a cup of tea. This list includes various utilities and libraries, the Apache web server, and “carto” which is used to

Installing postgresql / postgis

On Ubuntu there are pre-packaged versions of both postgis and postgresql, so these can simply be installed via the Ubuntu package manager.

```
sudo apt-get install postgresql postgresql-contrib postgis postgresql-10-postgis-2.
```

Here “postgresql” is the database we’re going to store map data and “postgis” adds some extra graphical support to it. Again, say yes to install.

Now you need to create a postgis database. The defaults of various programs assume the database is called gis and we will use the same convention in this tutorial, although this is not necessary. Substitute your username for renderaccount where is is used below. This should be the username that will render maps with Mapnik.

```
sudo -u postgres -i
createuser renderaccount # answer yes for superuser (although this isn't strictly n
createdb -E UTF8 -O renderaccount gis
```

While still working as the “postgres” user, set up PostGIS on the PostgreSQL database (again, substitute your username for renderaccount below):

```
psql
```

(that’ll put you at a “postgres=#” prompt)

```
\c gis
```

(it’ll answer “You are now connected to database ‘gis’ as user ‘postgres’”.)

```
CREATE EXTENSION postgis;
```

(it’ll answer CREATE EXTENSION)

```
CREATE EXTENSION hstore;
```

(it’ll answer CREATE EXTENSION)

(it'll answer ALTER TABLE)

```
ALTER TABLE spatial_ref_sys OWNER TO renderaccount;
```

(it'll answer ALTER TABLE)

```
\q
```

(it'll exit psql and go back to a normal Linux prompt)

```
exit
```

(to exit back to be the user that we were before we did "sudo -u postgres -i" above)

If you haven't already created one create a Unix user for this user, too, choosing a password when prompted:

```
sudo useradd -m renderaccount
sudo passwd renderaccount
```

Again, above replace "renderaccount" with the non-root username that you chose.

Installing osm2pgsql

We will need install various bits of software from source. The first of this is "osm2pgsql". Various tools to import and manage OpenStreetMap data into a database exist. Here we'll use "osm2pgsql", which is probably the most popular.

```
mkdir ~/src
cd ~/src
git clone git://github.com/openstreetmap/osm2pgsql.git
cd osm2pgsql
```

The build mechanism used by osm2pgsql has changed since older versions, so we'll need to install some more prerequisites for that:

```
sudo apt install make cmake g++ libboost-dev libboost-system-dev libboost-filesyste
```

Again, say yes to install.

(the output from that should end with “build files have been written to”)

```
make
```

(the output from that should finish with “[100%] Built target osm2pgsql”)

```
sudo make install
```

Mapnik

Next, we’ll install Mapnik. We’ll use the default version in Ubuntu 18.04:

```
sudo apt-get install autoconf apache2-dev libtool libxml2-dev libbz2-dev libgeos-dev
```

We’ll check that Mapnik has been installed correctly:

```
python
>>> import mapnik
>>>
```

If python replies with the second chevron prompt »> and without errors, then Mapnik library was found by Python. Congratulations! You can leave Python with this command:

```
>>> quit()
```

Install mod_tile and renderd

Next, we’ll install mod_tile and renderd. “mod_tile” is an Apache module that handles requests for tiles; “renderd” is a daemon that actually renders tiles when “mod_tile” requests them. We’ll use the “switch2osm” branch of https://github.com /SomeoneElseOSM/mod_tile, which is itself forked from https://github.com /openstreetmap/mod_tile, but modified so that it supports Ubuntu 18.04, and with a couple of other changes to work on a standard Ubuntu server rather than one of OSM’s rendering servers.

Compile the mod_tile source code:

```
cd mod_tile
./autogen.sh
```

(that should finish with "autoreconf: Leaving directory '.'".)

```
./configure
```

(that should finish with "config.status: executing libtool commands")

```
make
```

Note that some "worrying" messages will scroll up the screen here. However it should finish with "make[1]: Leaving directory '/home/renderaccount/src/mod_tile'".

```
sudo make install
```

(that should finish with "make[1]: Leaving directory '/home/renderaccount /src/mod_tile'")

```
sudo make install-mod_tile
```

(that should finish with "chmod 644 /usr/lib/apache2/modules/mod_tile.so")

```
sudo ldconfig
```

(that shouldn't reply with anything)

Stylesheet configuration

Now that all of the necessary software is installed, you will need to download and configure a stylesheet.

The style we'll use here is the one that use by the "standard" map on the openstreetmap.org website. It's chosen because it's well documented, and should work anywhere in the world (including in places with non-latin placenames). There are a couple of downsides though - it's very much a compromise designed to work globally, and it's quite complicated to understand and modify, should you need to do that.

The home of "OpenStreetMap Carto" on the web is https://github.com/gravitystorm /openstreetmap-carto/ and it has it's own installation instructions at

Here we're assuming that we're storing the stylesheet details in a directory below "src" below the home directory of the "renderaccount" user (or whichever other one you are using)

```
cd ~/src
git clone git://github.com/gravitystorm/openstreetmap-carto.git
cd openstreetmap-carto
```

Next, we'll install a suitable version of the "carto" compiler. This is later than the version that ships with Ubuntu, so we need to do:

```
sudo apt install npm nodejs
sudo npm install -g carto
carto -v
```

That should respond with a number that is at least as high as:

```
carto 1.1.0 (Carto map stylesheet compiler)
```

Then we convert the carto project into something that Mapnik can understand:

```
carto project.mml > mapnik.xml
```

You now have a Mapnik XML stylesheet at /home/renderaccount /src/openstreetmap-carto/mapnik.xml .

Loading data

Initially, we'll load only a small amount of test data. Other download locations are available, but "download.geofabrik.de" has a wide range of options. In this example we'll download the data for Azerbaijan, which is about 17Mb.

Browse to https://download.geofabrik.de/asia/azerbaijan.html and note the "This file was last modified" date (e.g. "2017-02-26T21:43:02Z"). We'll need that later if we want to update the database with people's susbsequent changes to OpenStreetMap. Download it as follows:

```
mkdir ~/data
cd ~/data
wget https://download.geofabrik.de/asia/azerbaijan-latest.osm.pbf
```

planet might take many hours, days or weeks depending on the hardware. For smaller extracts the import time is much faster accordingly, and you may need to experiment with different -C values to fit within your machine's available memory.

```
osm2pgsql -d gis --create --slim  -G --hstore --tag-transform-script ~/src/openstre
```

It's worth explaining a little bit about what those options mean:

```
-d gis
```

The database to work with ("gis" used to be the default; now it must be specified).

```
--create
```

Load data into an empty database rather than trying to append to an existing one.

```
--slim
```

osm2pgsql can use different table layouts; "slim" tables works for rendering.

```
-G
```

Determines how multipolygons are processed.

```
--hstore
```

Allows tags for which there are no explicit database columns to be used for rendering.

```
--tag-transform-script
```

Defines the lua script used for tag processing. This an easy is a way to process OSM tags before the style itself processes them, making the style logic potentially much simpler.

```
-C 2500
```

Allocate 2.5 Gb of memory to osm2pgsql to the import process. If you have less memory you could try a smaller number, and if the import process is killed because it runs out of memory you'll need to try a smaller number or a smaller OSM extract..

Use 1 CPU. If you have more cores available you can use more.

```
-S
```

Create the database columns in this file (actually these are unchanged from "openstreetmap-carto")

The final argument is the data file to load.

That command will complete with something like "Osm2pgsql took 238s overall".

Shapefile download

Although most of the data used to create the map is directly from the OpenStreetMap data file that you downloaded above, some shapefiles for things like low-zoom country bondaries are still needed. To download and index these:

```
cd ~/src/openstreetmap-carto/
scripts/get-shapefiles.py
```

This process involves a sizable download and may take some time. When complete it will display "…script completed.".

Fonts

The names used for places around the world aren't always written with latin characters (the familar western alphabet a-z). To install the necessary fonts do the following:

```
sudo apt-get install fonts-noto-cjk fonts-noto-hinted fonts-noto-unhinted ttf-unifo
```

OpenSteetMap Carto's own installation instructions also suggest installing "Noto Emoji Regular" from source. That is needed for the emojis in an American shop name, apparently. All the other international fonts that are likely to be needed (including ones often not supported) are including in the list just installed.

Setting up your webserver

Configure renderd

The config file for "renderd" is "/usr/local/etc/renderd.conf". Edit that with a text editor such as nano:

A couple of lines in here may need changing. In the "renderd" section:

```
num_threads=4
```

If you've only got 2Gb or so of memory you'll want to reduce this to 2. The "ajt" section corresponds to a "named map style" called "ajt". You can have more than one of these sections if you want, provided that the URI is different for each. The "XML" line will need changing to something like:

```
XML=/home/renderaccount/src/openstreetmap-carto/mapnik.xml
```

You'll want to change "renderaccount" to whatever non-root username you used above.

```
URI=/hot/
```

That was chosen so that the tiles generated here can more easily be used in place of the HOT tile layer at OpenStreetMap.org. You can use something else here, but "/hot/" is as good as anything.

Configuring Apache

```
sudo mkdir /var/lib/mod_tile
sudo chown renderaccount /var/lib/mod_tile

sudo mkdir /var/run/renderd
sudo chown renderaccount /var/run/renderd
```

We now need to tell Apache about "mod_tile", so with nano (or another editor):

```
sudo nano /etc/apache2/conf-available/mod_tile.conf
```

Add the following line to that file:

```
LoadModule tile_module /usr/lib/apache2/modules/mod_tile.so
```

and save it, and then run:

```
sudo a2enconf mod_tile
```

We now need to tell Apache about "renderd". With nano (or another editor):

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

And add the following between the "ServerAdmin" and "DocumentRoot" lines:

```
LoadTileConfigFile /usr/local/etc/renderd.conf
ModTileRenderdSocketName /var/run/renderd/renderd.sock
# Timeout before giving up for a tile to be rendered
ModTileRequestTimeout 0
# Timeout before giving up for a tile to be rendered that is otherwise missing
ModTileMissingRequestTimeout 30
```

And reload apache twice:

```
sudo service apache2 reload
sudo service apache2 reload
```

(I suspect that it needs doing twice because Apache gets "confused" when reconfigured when running)

If you point a web browser at: http://yourserveripaddress/index.html you should get Ubuntu / apache's "It works!" page.

(if you don't know what IP address it will have been assigned you can likely use "ifconfig" to find out - if the network configuration is not too complicated it'll probably be the "inet addr" that is not "127.0.0.1"). If you're using a server at a hosting provider then it's likely that your server's internal address will be different to the external address that has been allocated to you, but that external IP address will have already been sent to you and it'll probably be the one that you're accessing the server on currently.

Note that this is just the "http" (port 80) site - you'll need to do a little bit more Apache configuration if you want to enable https, but that's out of the scope of these instructions. However, if you use "Let's Encrypt" to issue certificates then the process of setting that up can also configure the Apache HTTPS site as well.

Running renderd for the first time

Next, we'll run renderd to try and render some tiles. Initially we'll run it in the foreground so that we can see any errors as they occur:

You may see some warnings here - don't worry about those for now. You shouldn't get any errors. If you do, save the full output in a pastebin and ask a question about the problem somewhere like help.openstreetmap.org (linking to the pastebin - don't include all the text in the question).

Point a web browser at: http://yourserveripaddress/hot/0/0/0.png

You should see a map of the world in your browser and some more debug on the command line, including "DEBUG: START TILE" and "DEBUG: DONE TILE". Ignore any "DEBUG: Failed to read cmd on fd" message - it is not an error. If you don't get a tile and get other errors again save the full output in a pastebin and ask a question about the problem somewhere like help.openstreetmap.org.

If that all works, press control-c to stop the foreground rendering process.

Running renderd in the background

Next we'll set up "renderd" to run in the background. First, edit the "~/src/mod_tile /debian/renderd.init" file so that "RUNASUSER" is set to the non-root account that you have used before, such as "renderaccount", then copy it to the system directory:

```
nano ~/src/mod_tile/debian/renderd.init
sudo cp ~/src/mod_tile/debian/renderd.init /etc/init.d/renderd
sudo chmod u+x /etc/init.d/renderd
sudo cp ~/src/mod_tile/debian/renderd.service /lib/systemd/system/
```

The "renderd.service" file is a "systemd" service file. The version used here just calls old-style init commands. In order to test that the start command works:

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

(that should reply with "[ok] Starting renderd (via systemctl): renderd.service".)

To make it start automatically every time:

```
sudo systemctl enable renderd
```

Viewing tiles

In order to see tiles, we'll cheat and use an html file "sample_leaflet.html" in mod_tile's "extras" folder. Just open that file in a web browser on the machine where you installed the tile server. If that isn't possible because you're installing on a

From an ssh connection do:

```
tail -f /var/log/syslog | grep " TILE "
```

(note the spaces around “TILE” there)

That will show a line every time a tile is requested, and one every time rendering of one is completed.

When you load that page you should see some tile requests. Zoom out gradually. You’ll see requests for new tiles show up in the ssh connection. Some low-zoom tiles may take a long time (several minutes) to render for the first time, but once done they’ll be ready for the next time that they are needed.

Congratulations. Head over to the using tiles section to create a map that uses your new tile server.