### Introduction

Mapproxy is a python proxy server for geospatial images. It can read data from WMS, tiles, MapServer and Mapnik, and cache and serve that data as [WMS](http://wiki.openstreetmap.org/wiki/WMS),[WMTS](http://www.opengeospatial.org/standards/wmts/),[TMS](http://wiki.openstreetmap.org/wiki/TMS) and [KML](http://wiki.openstreetmap.org/wiki/KML). It can also do reprojections between different coordinate reference systems.

You can click [here](http://mapproxy.org/docs/latest/install.html) to know how to install MapProxy.

### Configuration

MapProxy configuration is set up in the mapproxy.yaml file. The MapProxy configuration is a dictionay, each key configures a different aspect of MapProxy. There are the following keys:

* services: this is the place to activate and configure MapProxy’s services like WMS and TMS. They define how the tiles are organized.
* layers: configure the layers that MapProxy offers. Each layer can consist of multiple sources and caches. We can have several layers, in that case we need to use the transparent option.
* sources: define where MapProxy can retrieve new data, where our tiles are stored. One of the options will be the tiles url.
* caches: here you can configure the internal caches. When the layer is requested by a client, MapProxy looks in the cache for the requested data and only if it hasn’t cached the data yet, it requests the required data source.
* grids: MapProxy aligns all cached images (tiles) to a grid. Here you can define that grid. Among other options, you can define a factor between each resolution. It should be either a number or the term sqrt2. The default factor is 2.
* globals: here you can define some internals of MapProxy and default values that are used in the other configuration directives.

To start the development server we use the command:

mapproxy-util serve-develop mapproxy.yaml

### Seeding and caching

MapProxy creates all tiles on demand. That means, only tiles requested once are cached. Fortunately MapProxy comes with a command line script for pre-generating all required tiles called mapproxy-seed. It has its own configuration file called seed.yaml and a couple of options. In the file we can find these options.

* Seeds: configure seeding tasks. Here you can create multiple seeding tasks that define what should be seeded. You can specify a list of caches for seeding with caches. If you have specified multiple grids for one cache in your MapProxy configuration, you can select these caches to seed.
* Cleanups: configure cleanup tasks, to remove the old tiles stored. You can clean up caches, grids, levels, coverages, all tiles before a date.
* Coverages: configure coverages for seeding and cleanup tasks. You can define areas where data is available or where the interesting data is.

The tool can seed one or more polygon areas for each cached layer. It can be really useful for lake or sea areas, where the tiles are exactly the same to each others.

Apart from using normal files as cache, you can also use a single SQLite file for the cache. It uses the [MBTile specification](http://mbtiles.org/).

caches:

# name of our cache

mbtiles\_cache:

# A list of data sources for this cache. You can use sources defined in the sources and caches section. MapProxy will merge multiple sources from left (bottom) to right (top) before they are stored on disk.

sources: [my\_source]

# You can configure one or more grids for each cache. MapProxy will create one cache for each grid.

grids: [my\_grid]

# Configure the type of the background tile cache.

cache:

# You configure the type with thetype option. The default type is file

type: mbtiles

# This is where your .mbtiles file is located

filename: /path/to/cache.mbtiles

You could also set the sources to an empty list, if you use an existing MBTiles file and do not have a source.

You could also use a cache as the source of another cache. For example, you might need to change the grid of an existing cache to cover a larger bounding box, or to support tile clients that expect a different grid, but you do not want to seed the data again.

# Each layer contains information about the layer and where the data comes from.

layers:

# The name of the layer. You can omit the name for group layers.

- name: layer1

# Readable name of the layer, the one that will appear in the website.

title: Layer using data from existing\_cache

# A list of data sources for this layer. You can use sources defined in the sources and caches section. MapProxy will merge multiple sources.

sources: [new\_cache]

caches:

new\_cache:

grids: [new\_grid]

sources: [existing\_cache]

existing\_cache:

grids: [old\_grid]

sources: [my\_source]

# Here you can define the tile grids that MapProxy uses for the internal caching.

grids:

utm32n:

# The spatial reference system used for the internal cache, written as EPSG:xxxx.

srs: 'EPSG:27700'

# The extent of your grid. You can use either a list or a string with the lower left and upper right coordinates.

bbox: [4, 46, 16, 56]

# The SRS of the grid bbox.

bbox\_srs: 'EPSG:4326'

# The default origin (x=0, y=0) of the tile grid is the lower left corner, similar to TMS. WMTS defines the tile origin in the upper left corner. MapProxy can translate between services and caches with different tile origins, (but there are some limitations for grids with custom bbox and resolutions that are not of factor 2). ll or sw: if the x=0, y=0 tile is in the lower-left/south-west corner of the tile grid. This is the default. ul or nw: if the x=0, y=0 tile is in the upper-left/north-west corner of the tile grid.

origin: 'nw'

# The resolutions of the first level.

min\_res: 5700

osm\_grid:

# With this option, you can base the grid on the options of another grid you already defined.

base: GLOBAL\_MERCATOR

origin: nw

### Grids

By default the resolutions between each pyramid level doubles. If you want to change this, you can do so by [defining your own grid](http://mapproxy.org/docs/nightly/configuration.html#id5). Fortunately MapProxy grids provied the ability to inherit from another grid. We let our grid inherit from the previously used GLOBAL\_GEODETIC grid and add five fixed resolutions to it.

grids:

res\_grid:

base: GLOBAL\_GEODETIC

# A list with all resolutions that MapProxy should cache.

res: [1, 0.5, 0.25, 0.125, 0.0625]

The resolutions are always in the unit of the SRS, in this case in degree per pixel.

Instead of defining fixed resolutions, we can also define a factor that is used to calculate the resolutions. The default value of this factor is 2, but you can set it to each value you want. Just change res with res\_factor and add your preferred factor after it.

A magical value of res\_factor is sqrt2, the square root of two. It doubles the number of cached resolutions, so you have 40 instead of 20 available resolutions.

Let see how to [define our own grid](http://mapproxy.org/docs/nightly/configuration.html#id5).

For this example we define a grid for Germany. We need a spatial reference system (srs) that match the region of Germany and a bounding box (bbox) around Germany to limit the requestable aera. To make the specification of the bbox a little bit easier, we put the bbox\_srs parameter to the grid configuration. So we can define the bbox in EPSG:4326 (European Petroleum Survey Group).

germany:

srs: 'EPSG:25832'

bbox: [6, 47.3, 15.1, 55]

bbox\_srs: 'EPSG:4326'

In bbox we consider: lat\_min,lon\_min,lat\_max,lon\_max; which means minimal latitude is 6, minimal longitude is 47.3, maximal latitude is 15.1, maximal longitude is 55.

### Defining Resolutions

There are multiple options that influence the resolutions MapProxy will use for caching: res, res\_factor, min\_res, max\_res, num\_levels and also bbox andtile\_size.

If you supply a list with resolution values in res then MapProxy will use this list and will ignore all other options.

If min\_res is set then this value will be used for the first level, otherwise MapProxy will use the resolution that is needed for a single tile (tile\_size) that contains the whole bbox.

If you have max\_res and num\_levels: the resolutions will be distributed between min\_res and max\_res, both resolutions included. The resolutions will be logarithmical, so you will get a constant factor between each resolution. For example, with resolutions from 1000 to 10 and 6 levels you would get 1000, 398, 158, 63, 25, 10 (rounded here).

If you have max\_res and res\_factor: the resolutions will be multiplied by res\_factor until larger then max\_res.

If you have num\_levels and res\_factor: the resolutions will be multiplied by res\_factor for up to num\_levels levels.

You can set every cache resolution in the res option of a layer.

caches:

custom\_res\_cache:

grids: [custom\_res]

sources: [vector\_source]

grids:

custom\_res\_cache:

srs: 'EPSG:31467'

res: [10000, 7500, 5000, 3500, 2500]

You can specify a different factor that is used to calculate the resolutions. By default a factor of 2 is used, but you can set smaller values:

grids:

custom\_factor:

# Here you can define a factor between each resolution. It should be either a number or the term sqrt2.

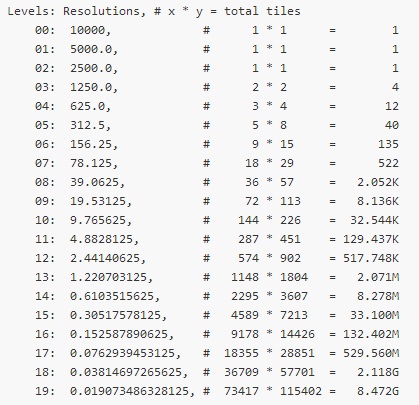
res\_factor: 1.6

Other option is a convenient variation of the previous option. A factor of 1.41421, the square root of two, would get resolutions of 10, 7.07, 5, 3.54, 2.5,…. Notice that every second resolution is identical to the power-of-two resolutions. This comes in handy if you use the layer not only in classic WMS clients but also want to use it in tile-based clients like OpenLayers, which only request in these resolutions.

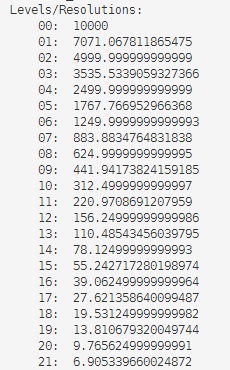
grids:

sqrt2:

res\_factor: sqrt2



In this example, we have only chosen min\_res: 10000; and the res\_factor will be 2, which is the default value. And bbox: [5, 50, 10, 55]



Here we select min\_res: 10000 and res\_factor: ‘sqrt2’

### Reprojecting

When the grids are not compatible, for example, when they use different projections, then MapProxy will access the source cache as if it is a WMS source and it will use meta-requests and do image reprojection as necessary.

It will reproject data if it needs to get data from this layer in any other SRS.

If MapProxy needs to reproject and the source has multiple supported\_srs, then it will use the fist projected SRS for requests in projected SRS, or the fist geographic SRS for requests in geographic SRS. For example, when supported\_srs is ['EPSG:4326','EPSG:31467'] caches with EPSG:900913 will use EPSG:32467.

Here is an example that uses OSM tiles as a source and offers them in UTM projection. The disable\_storageoption prevents MapProxy from building up two caches. The meta\_size makes MapProxy to reproject multiple tiles at once.

Note that reprojecting vector data results in quality loss. For better results you need to find similar resolutions between both grids.

layers:

- name: osm

title: OSM in UTM

sources: [osm\_cache]

caches:

osm\_cache:

grids: [utm32n]

# MapProxy does not make a single request for every tile but will request a large meta-tile that consist of multiple tiles. meta\_size defines how large a meta-tile is. A meta\_size of [4, 4] will request 16 tiles in one pass. With a tile size of 256x256 this will result in 1024x1024 requests to the source WMS.

meta\_size: [4, 4]

sources: [osm\_cache\_in]

osm\_cache\_in:

grids: [osm\_grid]

# If set to true, MapProxy will not store any tiles for this cache. MapProxy will re-request all required tiles for each incoming request, even if the there are matching tiles in the cache.

disable\_storage: true

sources: [osm\_source]

# You need to choose a unique name for each configured source. This name will be used to reference the source in the caches and layers configuration.

sources:

osm\_source:

# Use the type tile to request data from from existing tile servers like TileCache and GeoWebCache.

type: tile

# The grid of the tile source. Defaults to GLOBAL\_MERCATOR, a grid that is compatible with popular web mapping applications.

grid: osm\_grid

# This source takes a url option that contains a URL template. The template format is %(key\_name)s.

url: http://a.tile.openstreetmap.org/%(z)s/%(x)s/%(y)s.png

grids:

utm32n:

srs: 'EPSG:25832'

bbox: [4, 46, 16, 56]

bbox\_srs: 'EPSG:4326'

origin: 'nw'

min\_res: 5700

osm\_grid:

base: GLOBAL\_MERCATOR

origin: nw

If you do not want to cache data but still want to use MapProxy’s ability to reproject WMS layers on the fly, you can use a direct layer. Add your source directly to your layer instead of a cache.

You should explicitly define the SRS the source WMS supports. Requests in other SRS will be reprojected. You should specify at least one geographic and one projected SRS to limit the distortions from reprojection.

layers:

- name: direct\_layer

sources: [direct\_wms]

sources:

direct\_wms:

# Use the type wms to for WMS servers.

type: wms

# A list with SRSs that the WMS source supports. MapProxy will only query the source in these SRSs. It will reproject data if it needs to get data from this layer in any other SRS.

supported\_srs: ['EPSG:4326', 'EPSG:25832']

# This describes the WMS source. The only required options are url and layers. You need to set transparent to true if you want to use this source as an overlay.

req:

url: http://wms.example.org/service?

layers: layer0, layer1