Osmhike

Hiking oriented TileServer based on Cyclosm

Beginner's tutorial

Bernard Maison 13th May 2025

Contents

1 Introduction

Osmhike is a Linux/Docker based implementation of an Openstreetmap Tile Server. It uses Openstreetmap data.
It has been tuned for hiking activity, but also covers some cycling needs since it is derived fromCyclosm.

1.1 Why a Cyclosm adapted clone ?

The goal is double:

- have a personal tile-server, that will allow to produce personal maps, adapted to mountain hiking.
- have a complete tutorial, allowing beginners to discover and understand the underlying magics

Several solutions were possible:

- Openstreetmap, but it does not contain contours lines and hillshade
- Opentopomap and Cyclosm both contain contours lines and hillshade

The final choice was Cyclosm, for 2 reasons

- the yaml format for defining map style is **much** more confortable than the xml files used by Opentopomap
- the software infrastructure based on Kosmtik (tile server) tool is **much** simpler than Opentopomap

But Cyclosm could not be used directly (cyclosm-cartocss-style-master on github sept2024), because the stuff relative to contours/hillshade did not work:

- missing scripts
- incomplete documentation to implement them

Cyclosm brings a lot of stuff for the needs of cycling activity. This causes some complexity.
This was left nearly unchanged . But some changes have been made, for a better adaptation to mountain hiking

- Use flashy colors (red,yellow) for primary/secondary roads (like in road maps)
- Lighten colors for landuse (forest, meadow , heath/scrub) so that mountain paths are more visible
- Unify track/bridleway/footway : same linedash/size style , just keep different colors
  Use different linedash to mark the compatibility with biking
- Use flashy colors for path , so that mountain path become well visible.
  Make color distinction for path usable by cyclocross/mountainbike
  Use different linedash patterns to mark alpine and difficult path
- Simplify protected areas
- Remove (commented out) blocks to indicate MountainBike difficulty
- Enlarge the overlay corresponding to cycling routes, to avoid mysterious changes in road colors.
  Use only 3 colors : (mountainbike,local,not local) to avoid managing too many colors
- Remove most administrative limits, which bring useless pollution
- Make elevation data more readable, on contour lines

Finaly, implementation was completely reworked

- Move to more recent OS version (ubuntu:noble, then debian:bookworm for better stability)
- Replace obsolete Phyghtmap by Pyhgtmap for contours generation
- Various improvements, and strong simplification of stylesheet for roads
- Change Docker implementation
- Add plenty of comments everywhere

Kosmtik or Tirex ?
The use of Kosmtik was promoted by Cyclosm, for implementing the tile server
Tirex is also proposed as an alternative solution, but with a **mixed** solution : (yaml format for the map style, Tirex for tile server)

1.2 The notion of Spatial Reference System (SRS)

Why is it necessary to get a mimum understanding of this concept ?
Because working with maps and Openstreetmap data, we are continuously bothered with it !

A Spatial Reference System is a model used to measure locations on Earth

Some of them give a quite exact representation of reality

- "Geographic/Geodetic coordinate system" : longitude/latitude/elevation coordinates
- "Geocentric/Earth centered coordinate system" : x / y / z cartesian coordinates, relative to an origin (Earth center/equatorial plane/0° meridian)

Some others give an approximative representation, because the are "projections" to a flat map

- "Projected coordinate system" : x,y location on a plan map , relative to an arbitrary (0,0) origin

Things are complicated by the fact that Earth is not a pure sphere. It's rather an ellipsoid flattened on the poles.
This has a very complex impact on the method to determine Latitude...

Each SRS is given an "EPSG number" , provided by **European Petroleum Survey Group**

Two SRS are particularly important for us

WGS84 (EPSG:4326)	World Geodedic System 1984 The major standard, used by GPS positionning longitude/latitude coordinates, considering Earth a an ellipsoid, based on Earth center and equatorial plane
Web-Mercator Pseudo-Mercator Spherical-Mercator (EPSG:3857)	many names for this special Mercator projection, used by Google and Openstreetmap uses longitude/latitude values coming from WGS84, but projects them to X,Y coordinates as if Earth was a perfect sphere (this simplification is made because it needs much less computations)

Concretely :

- Openstreetmap data containing longitude/latitude coordinates are defined according to EPSG:4326
  When making maps, the rendering process must be told to convert them towards EPSG:3857
- Some Elevation (altitude) data from Nasa also follow EPSG:4326 . To produce a .tif flat image, compatible with Openstreetmap maps, we must explicitely tell the tools to apply a convertion towards EPSG:3857

1.3 Prerequisites

Docker installation

You must install Docker on your Host

Happily, modern Linux distributions have docker available in their standard repositories
So installation is not more complex than: (Ubuntu/Debian)
apt update && apt install docker.io docker-compose

Well, official docker documentation tells that this is older versions, and that you should rather use their uptodate version called "docker-ce" (which needs to make some complex manipulations with repositories and certificates)

Don't know Docker ? There is a mini tutorial at the end of this document

Required knowledge

If you are trying to build a tile server, I guess that you know what a tile server is...

You should know a minimum concerning :

- SQL database language
- YAML syntax

2 OSM Data

2.1 Download OSM data

There are 3 main file formats

.osm	XML file .
.osm.bz2	Compressed XML file
.osm.pbf	Binary format, more compact than xml This is the preferred format to use with tool osm2pgsql

In some occasions, polygon files may be useful

.poly	Text file containing a list of (longitude,latitude) points defining the border of a country or region

An easy way to download osm files is https://download.geofabrik.de/ which allows to navigate into world regions / countries / sub regions

A file looks like france-latest.osm.pbf

To have a direct access to all the files proposed for France try : http://download.geofabrik.de/europe/france/

Note:
There are tools or web sites, that allow to build the .osm file corresponding to a specific area

2.2 OSM file structure

Here we will have a look at a the XML file structure to understand the type of objects it contains

There are 3 main types of objects

node	It describes a Point with longitute/latitude coordinates
way	Describes either a Line (road, river ...) of a Polygon (forest area , lake ...) , defined by a list of node elements A Polygon is just a «way» whose first and last nodes are equal *Note: yes, they could have chosen a better name for this concept ... but it is due to historical reasons*
relation	Allows to group objects - a bus line is a set of several roads - a lake containing an island may be defined by 2 «way» objects: the outer way that describes the lake border, the inner way that describes the island

Those objects have some common attributes, the most important ones being

id	Unique identifier

tag	Each object may have several tags, which define
	• what it is (a town, a road, a river, a lake, a building …)
	• its characteristics (population, name , usage …)
	
	A tag is a pair (name,value) like :
	`<tag k="highway" v="path"/>`
	`<tag k="name" v="Mt Blanc"/>`
	
	The list of possible tag names is normalized.
	For complete list and description, see :
	https://wiki.openstreetmap.org/wiki/Map_features

What is important to know when making maps for hiking, is that both roads and little paths in the mountain are classified with the same tag name «highwa » . The tag value gives the exact nature

Most important values in this case are

Tag value	**Description**
motorway trunk	motorway
primary secondary tertiary	different kinds of roads with declining importance
track	Roads for mostly *agricultural or forestry usage*
path	path in the mountain
cycleway	for bicycles

Example: node

```
<node id="45160958" version="19" lat="42.6037228" lon="1.4419909" >
   <tag k="ele" v="2914"/>
   <tag k="name" v="Pic de Médécourbe"/>
 </node>
```

Example: way
the <nd ref=xxxxx/> topics give the list of nodes (using their id) that compose the way

```
<way id="6179270" version="11" timestamp="2023-03-13T07:10:03Z">
   <nd ref="51371386"/>
   <nd ref="10732120402"/>
   <nd ref="51384490"/>
    <tag k="highway" v="secondary"/>
   <tag k="surface" v="asphalt"/>
 </way>
```

More details https://wiki.openstreetmap.org/wiki/OSM_XML https://wiki.openstreetmap.org/wiki/Elements

2.3 Shapefile

This is a complex format to describe geographical objects, composed of at least 4 files :

.shp	position and form of the objects
.dbf	database containing the attributes
.shx	indexes
.prj	description of the coordinate system

Shapefile is often used to distribute data. Many tools can directly show a shapefile
We generally download them as a .zip file

Why mentionning this format in this tutorial ?
Just because the description of 'Water Polygons' (seas/oceans borders) , used to render the world map at low zoom, is available in this format (instead of a standard .osm file) !

3 Understanding PostGIS

PostGIS is an essential point of the global architecture

It is an extension for the Postgresql database, that allows to manage geographical data

This means:

- define a new datatype GEOMETRY, and allow the database to create columns with this datatype
- provide plenty of functions to insert,manipulate this kind of data
  example:
  # returns the list of points common to object1,object2
  `ST_Intersection( object1, object2 )`

The main subtypes of GEOMETRY are

Point	Point defined by its longitude/latitude coordinates
LineString	A curve defined by a set of Points
Polygon	A "closed" Line forming an area because first Point equals last Point
MultiPolygon	Several Polygons (example: a lake with an island inside)
MultiLineString	a collection of Lines
MultiPoint	a collection of Points

There is also the notion of "BoundingBox" associated to a GEOMETRY object (named "bbox" by some tools) : the smallest rectangle – parallel to the coordinate axes – capable of containing the object

Installation:
The extension must be added to each Postgresql database that you create
psql -d mydatabase -c 'CREATE EXTENSION IF NOT EXISTS postgis;'

Note:
Another useful extension is "hstore", which allows to efficiently manage (key,value) pairs in a column
psql -d mydatabase -c 'CREATE EXTENSION IF NOT EXISTS hstore;'

More details
http://postgis.net/workshops/postgis-intro/introduction.html
http://postgis.net/workshops/postgis-intro/

4 Loading data with osm2pgsql

osm2pgsql is a tool that will load osm file xxxx.osm.pbf into a PostGIS database

4.1 PostGIS database structure

Why is it necessary to understand the structure of the database generated by osm2pgsql ?
Because you will use SQL requests to extract data, for rendering tiles !

The correspondance between elements in the XML data , and database tables is:

TABLE	XML data
planet_osm_point	populated from <node> XML elements
planet_osm_line	populated from <way> XML elements
planet_osm_polygon	populated from "closed" <way> and some <relation> XML elements *it seems that osm2pgsql converts <way> elements to polygon only if some attributes like "natural" or "landuse" are present....*
planet_osm_roads	some specific <way> elements, whose tag "highway" tell there are roads, track , path ....

Those tables have almost the same structure :

- a column for each tagname (containing the tag value)
- a column "way" of type GEOMETRY
  (yes ! even for <node> elements)
- some other technical columns
  - z_order :
    INT4 column that may be used for ordering objects in the render. It mostly applies to objects with highway=* or railway=*
  - way_area :
    gives the area (square meters/feet) of the object, when relevant
    this is used in some cases, to decide whether the object is sufficiently big to be drawn

What is the exact type of this column "way" ?
A special technical PostGIS view, keeps track of all GEOMETRY columns in all tables

SELECT * FROM geometry_columns;

```
f_table_catalog | f_table_schema |      f_table_name       | f_geometry_column | coord_dimension | srid |    type
-----------------+----------------+-------------------------+-------------------+-----------------+------+------------
 osm             | public         | planet_osm_polygon      | way               |               2 | 3857 | GEOMETRY
 osm             | public         | planet_osm_point        | way               |               2 | 3857 | POINT
 osm             | public         | planet_osm_roads        | way               |               2 | 3857 | LINESTRING
 osm             | public         | planet_osm_line         | way               |               2 | 3857 | LINESTRING
 osm             | public         | cyclosm_ways            | way               |               2 | 3857 | LINESTRING
 osm             | public         | cyclosm_amenities_point | way               |               2 | 3857 | POINT
 osm             | public         | cyclosm_amenities_poly  | way               |               2 | 3857 | GEOMETRY
```

for planet_osm_polygon, we see "GEOMETRY" without detail : depending on the case, it is "Polygon" or "MultiPolygon"

Note that cyclosm has defined some specific views "cyclosm_xxxxx" to gather specific cycling stuff

More details
https://osm2pgsql.org/doc/manual.html#geometry-processing

4.2 .style config file

Well, this name is quite confusing ... This does not describe the rendering style (the colors to use in the final map ...) but gives indications about how to transform tags from XML input file, to columns in the database
osm2pgsql uses a default file: /usr/share/osm2pgsql/default.style

It is a text file, whose contents looks like:

```
node,way   harbour     text       polygon
node       ele         text       linear
.....
```

This means:

- create a column "harbour" of type TEXT , in the tables populated from XML elements , when <node> or <way>is found
  (put in this column, the values corresponding to tagname="harbour')
- same operation for column "ele" , but is affects only <node> elements

Note that this file also allows to force the creation of special columns "way_area" and "z_order" which are not tags found in the XML data . Their value is computed during processing

More details

- see explanations in the header of the standard file /usr/share/osm2pgsql/default.style which comes with the osm2pgsql package
- https://osm2pgsql.org/doc/manual.html#the-pgsql-output (chapter Style File)
- https://osm2pgsql.org/doc/man/

Note:
Normally, you don't have to bother with this file.
But for inserting elevation countours, it was necessary to provide a .style file with adequate contents (see later in this tutorial ...)

4.3 More details

See inside file docker-startup.sh the options used to call osm2pgsql

We use -s option (slim mode) and and --drop (clean temporary stuff when finished) :
Temporary stuff is written in database, rather than using memory (otherwise process will fail if you don't have enough memory) This means that database size will grow a lot during the process, then decrease a lot ...

https://osm2pgsql.org/doc/man/

5 Rendering configuration

This involve 2 steps:

- project.mml: extract adequate data from the database
- *.mss files : apply a style (colors, line thickness, labels font ...)

The final map is made of several "layers" which are drawn one upon the others. So the order in project.mml is important. It is also possible to add some transparency to a layer, so that it does not completely cover the former layers.

The order of layers is typically (approximative description)

landuse	shows the usage of land: forest, meadows ...
hillshade	draws shadow to give a visual image of topography
contour	draws contour lines (lines of points having same altitude)
waterway	rivers and lakes
tunnel and turning circle	
road, track, path	
aerial way, bridge	
administrative limit	country,department borders ...
ferry route	
bicycle route	emphasis made on itineraries for bicycles
GPS route	obtained from a dedicated file containing GPS route
electric power line	
protected area	limits of national/regional park...
tree	draw a specific icon to show the kind of tree inside a forest
label	names : country , city , protected area, road , bicycle route ...
amenity	the icons for different kinds of "point of interest" : picnic table, water see the attribute "amenity" in OSM documentation
amenity label	the text associated with the icon

The same kind of object, is often rendered by several layers, each of them covering a specific range of zoom

It is possible to define properties for a layer-name

- minzoom and maxzoom values : The layer will be drawn only if current zoom matches
- status: off allows to remove the rendering of the layer (without having to comment out the full description)

For example, concerning roads

Layer	Description
roads_low	zoom levels 7 to 8 railways , motoways & trunks
roads_med	zoom levels 9 to 10 railways roads having "highway attribute" IN ('motorway', 'trunk', 'primary', 'secondary')
roads_high	zoom >= 11 all railways, roads & track & path

5.1 project.mml

This file follows YAML syntax
It describes layers , and how to extract their data from database

Each layer is associated with a SQL request, with the following rules:

- it must return a TABLE
  so the standard syntax is : (SELECT .... FROM sometable) AS mytablename
  the name "mytablename" is not important, you can choose the one you like, and even use the same name for each layer
- the TABLE must contain an attribute named "way" which is a GEOMETRY object . Normally, it should be the attribute "way" from planet_osm_point, planet_osm_line ....
- it can also contain some extra attributes (either extracted from database, or computed), that will be used for final rendering

It contains 3 main sections

5.1.1 section "_parts"

It defines blocks of constants, that can be reused later in other sections

For example, the following syntax, declares a block named "osm2pgsql" which defines values for "type" and "dbname"

```
osm2pgsql: &osm2pgsql
  type: "postgis"
  dbname: "osm"
```

Later in the document, this syntax allows to inject keys "type" , "dbname" and their value

```
<<: *osm2pgsql
```

5.1.2 section Stylesheet

It lists the .mss stylesheet files, that be will used for rendering

5.1.3 section Layer

It is a collection of blocks that define the layers to be extracted from database, and drawn
- id: layer-name1
- id: layer-name2
....
the contents of a layer looks like

<<: *srs-osm	gets the keywords defined in block "srs-osm" This is the definition of the Spatial Reference System (SRS) known as "web-mercator" projection (the one used by google and osm to transform longtitude/latitude coordinates to a planned map) This projection considers the Earth as a perfect sphere, and not an ellipsoid (poles are indeed a little flattened) . This simplification makes computations faster ...

Datasource:
  <<: *osm2pgsql
  table: |-
  (SELECT
  .....
  FROM ….
  WHERE ....
) AS data
	<<: *osm2pgsql gets the keywords defined in block "osm2pgsql"
	• the name of database , and the fact that the column containing the GEOMETRY object is "way"
	• the "extent" key, which defines the bounds of the maps (tiles outside those bounds will not be rendered)
	
	the syntax |- is a funny YAML syntax telling that what follows is a multiline block
	
	the SQL request "(SELECT ....) AS data" extracts data from the database

geometry: polygon	tells the kind of GEOMETRY object (point, line,polygon ...)

properties:
  minzoom: 2
  maxzoom: 9
  status: off
	this block allows to define minzoom and maxzoom that apply to this layer
	
	leaving "status: off" will prevent this layer from being shown

5.2 *.mss files

Those files have CSS syntax [with some adaptations] . They follow the CartoCSS specification

5.2.1 Basic syntax

#buildings { style-definition }	applies to layers defined by id: buildings
.contours { style-definition }	applies to layers having the declaration in project.mml: class: contours
#waterway_low[zoom>=8][zoom<=12] { line-color: blue; [zoom=8] { line-width: 0.1; } [zoom=9] { line-width: 0.2; }	this block applies only if zoom is inside [8,12] defines line-color for all cases defines specific line-width with depending on zoom

[sport='cycling'][zoom>=15], [sport='bmx'][zoom>=15]
{
style-definition
}
	looks at the value of column named "sport", and applies if value is "cycling" OR "bmx"

@wooded: #d0feb9;	defines a constant named "wooded" that an be reused later. obviously this is a RGB color definition
[type='landuse_forest'] { polygon-fill: @wooded; }	recovers the value of the constant @wooded which was formely defined

#capital-names {
[zoom >= 8] {
text-name: '[name]';
}
}
	find out the name of capitals inside column "name" , but prints it only if zoom>=8

5.2.2 Drawing several time the same layer

It is possible to to draw several times a same layer, using different styles values.

This is typically used for "road-casing", that is drawing a border using a different color on the sides on a road

First solution using "::"

#bigroad::outline {
line-color: black;
line-width: 4;
}
 	
#bigroad::inline {
line-color: red;
line-width: 2;
}
	draws a large 4 pixels black line , and after draws again as a small 2 pixels red line
	
	the result is a red line with a black border

Second solution using "/"

#bigroad {
line-color: black;
line-width: 4;
 	
casing/line-color: red;
casing/line-width: 2;
}	

note that you can freely choose the name of the keyword after :: or before /

5.2.3 Rendering labels

It is quite touchy !

There are three possibilities:

- SHIELD : A .svg icon . Text is written inside the icon, or at a specific distance (dx, dy)
  Can be resized with complex syntax (shield-transform: "scale(0.75)";)
  Icon color is fixed, but text color can change (shield-fill)

- TEXT : only the label is printed . At the exact point, or at a specific distance (dx, dy)

- MARKER: A .svg icon . Can be resized easyly (marker-width,marker-height) . Can also modify color. Does not contain text. It's your job to draw a separate TEXT object

When dealing with labels, we may encounter many potential troubles:

- by default, those labels must not overlap: in this case, one of them will disappear !
  (this can cause strange effects, like the name of a mountain disappearing at a specific zoom, just because of overlapping trouble with other markers)
- a SHIELD using dx,dy may use much more room than just its text, increasing the risk of overlapping
- when using a tupple (MARKER, TEXT), one of them may disappear, because they have a separate overlapping management
- some strange bugs when a text overlaps a tile boundary (part of the text clipped, or text repeated at different places)
  To avoid this, we may add a key "buffer-size: 128" at the top of project.mml, to define an extra buffer around the tiles to render. If a text overlaps tile boundary, there is a chance that the text fits in the extra buffer, and is rendered correctly. *https://tilemill-project.github.io/tilemill/docs/guides/metatiles/*
  For a 256*256 pixels tile, "buffer-size: 128" means there are 64 extra pixels at each side of the tile

5.2.4 Hints

dashed lines

line-dasharray allows to draw dashed lines

It is important to add those parameters:

- line-cap: butt; // line terminates abruptly, without extra termination pixels
- line-join: round; // smooth rounded junction between joining lines

Otherwise, if you want to draw a line as a succession of little points, there is the risk that they look like a continuous line, because of added termination stuff.

magic tokens

Some magic tokens can be used in SQL requests, in conjunction with 'way_area' :
Those tokens are processed by the Mapnik renderer

!scale_denominator!	for a 1/17000 scale, this value contains 17000 (one cm on the device showing the map, equals scale_denominator cm in real world) *This is highly dependent on the pixel size of the device* *Mapnik seems to use a standard pixel size value of 0.28 milimeter : (90.71 dot per inch)* *Due to distance distorsion caused by Mercator projection, the scale is supposed to vary depending on Latitude.* *But, for a defined zoom value, Mapnik uses the same scale_denominator value whatever the latitude*
!pixel_width! !pixel_height!	tells how many meters a map pixel represents in reality Use them with extreme caution !! (their value may be always 0)

```
-- Compute a column 'way_pixels' telling how many pixels a polygon uses in the map .
-- Can be used to eliminate small objects
SELECT way_area/NULLIF(!pixel_width!::real*!pixel_height!::real,0) AS way_pixels

-- Since there is the risk that pixel_width is always 0 , we can replace them by an equivalent formula:
-- pixel_width = pixel_height= 0.00028 * scale_denominator
-- 1 meter on device represents  scale_denominator  meters in real world
-- so a pixel which is 0.00028 meters on device represents 0.00028 * scale_denominator in real world
```

Unfortunately, there is no !zoom! token to get current zoom value, but it can be retrieved by the formula:
ROUND (log(2 , 559082264 / !scale_denominator!))

magic token & performance

Another magic token is:
AND way && !bbox!
which means select only objects that fit in the area to draw.
This reduces the amount of objects to extract from database

it will be converted to something like
AND way && ST_SetSRID('BOX3D(some coordinates)'::box3d, 3857)

sql transformation

The sql inside project.mml is not exactly the one finally executed...

Generally, the sql request (SELECT ....) AS data will be suffixed by something like
WHERE "way" && ST_SetSRID('BOX3D(some coordinates)'::box3d, 3857)
which uses a postgis builtin function, to select only objects that fit in the area to draw.

This does not happen if the magic keyword *!bbox!* is found inside the request (or in comments !)

Caution!
Do not comment out the sequence "*AND way && !bbox!"* .
Since keyword !bbox! will still be found, the request will not be suffixed, causing a huge extraction of data

Generally both methods are equivalent (magic sequence !bbox! , automatic addition of suffix) , because the index filtering on object coordinates is done very early...

Exception : when column "way" is recomputed (ST_Snaptogrid(way, !pixel_width! / 4) AS way)
In this case, with automatic addition, the filtering on object coordinates will be done much later
So, the addition of magic sequence !bbox! is necessary

For example, the rendering of layer landuse-low , without !bbox! , brings a high risk of "shared memory error"

cache-feature

When a layer is drawn several times (see "casing" in previous chapter) , the rendering tool Mapnik has the default behavior to reexecute the same request several times !
Adding "cache-feature: true" in section properties: will avoid this
(the result of sql request will be cached in memory . of course this has a memory cost ...)

rendering priority

Layers and data are rendered according to their order of appearance.
- the first layers in project.mml are rendered before the others
- the first objects returned by a SQL request are rendered before the others

If a style (like line-width:) appears several times with different values:
- the last value will be used
- exception : with casing (using :: or / syntax) drawing is done twice

There is a special behaviour for objects rendered by text/marker/shield
For them, the standard option is to forbid overlapping: this means that the first rendered objects may prevent other ones from being drawn if they overlap with the first ones (even if positionned in different layers) ...

5.3 More details

project.mml and *.mss files follow the CartoCSS specification, which has been adopted by Openstreetmap.
But internally, a tool named "carto" transforms them into XML files, which are the ones expected by the rendering engine named "Mapnik"

It is not easy to find this CartoCSS full specification .
One solution is to look at documentation of other tools, (sometimes obsolete) which use it.

https://tilemill-project.github.io/tilemill/docs/manual/
https://tilemill-project.github.io/tilemill/docs/guides/styling-lines/
(introduction and very nice advanced recipes)

http://mapnik.org/mapnik-reference/#3.0.22
(a view of all possible keywords .
(Pay attention that the documented format *key=value* **must** be adapted to .CSS language)

https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#Resolution_and_Scale
(scale computation)

https://github.com/mapbox/carto/blob/master/docs/language_elements.rst
https://github.com/mapbox/carto/blob/master/docs/styling_concepts.rst

https://github.com/mapbox/mapbox-studio-classic/blob/mb-pages/docs/studio-classic-manual/04-classic-manual-cartocss.md

Zoom computation formulas

At zoom 0 , the world map is represented by a 256 pixels wide map
Circumference at equator : 40075km
At zoom z, the world map is represented by a 256 * 2^z pixels wide map
Size of a pixel on rendering device is supposed to be 0.28 mm

40075 * 1000 meters on earth correspond to 256 * 2^z * 0.28 * 0.001 meters on device
scale_denominator meters on earth corresponds to 1 meter on device

scale_denominator = 40075*1000 / (256 * 0.00028 * 2^z) = 559082264 / 2^z
2^z = 40075*1000 / (256 * 0.00028 * scale_denominator)
z = log2 [40075*1000 / (256 * 0.00028 * scale_denominator)]
z= ROUND (log2(559082264 / scale_denominator))

6 Drawing contours lines

This means drawing lines showing points of same altitude

This implementation uses two directories:

- dem : contains scripts and parameters
- demdata : working directory for downloaded files , and output files

6.1 DEM data from SRTM

DEM means "Digital Elevation Model" and DEM data refers to files containing the altitude of points on Earth
SRTM (Shuttle Radar Topography Mission) produced by Nasa, provides 2 levels of precision

- srtm3 : 3 second arc precision . This is 100m . Really insufficient
- srtm1 : 1 second arc precision . This is 30m

Data are .hgt files . Each file covers a square of 1 degree longitude and latitude.

A .hgt file is named like N45E50.hgt (45° longitude North , 50° latitude East) containing the altitudes as 16bits integers (and off course a header describing the contents)

srtm1 (one arc minute) files provide 3601x3601 values
srtm3 (3 arc minute) files provide 1201x1201 values

srtm-version is something completely different: it refers to the different shuttle missions
Current version is 3 . For this version, srtm1 dataset is not limited to US, and "holes" have been completed

Downloading files from Nasa is not the best solution:

- it requires a complex registration, to get login/pwd
- you may encounter "not authorized" errors

Happily, there is a mirror site : https://viewfinderpanoramas.org
which allows to download them easily (and it seems that for France, data does not come from Nasa, but from other method with higher quality (LIDAR))

6.2 pyhgtmap

This magic tool does all the stuff

- you give it a polygon file (.poly) giving the limits of the area you want
- it downloads the .hgt files either from Nasa or viewfinderpanorama that correspond to this area
- it puts them in a cache directory, so that next time you launch the tool, it can retrieve already existing files
- it builds the contour lines into a .pbf file, usable by osm2pgsql

Invocation:

pyhgtmap.sh --polygon=area.poly -j 2 -s 10 -0 --source=view1 --max-nodes-per-tile=0 --max-nodes-per-way=0 --pbf -o PREFIX
(pyhgtmap.sh is a personnal encapsulation ... see Installation)

--polygon	tells the .poly file defining the polygon of the area to process
-j 8	runs 8 jobs in parallel
-s 10	contour line step size in meters Phyghtmap will produce only contours lines whose "ele" is multiple of 10
-0	do not produce contours for altitude=0 (avoid troubles with oceans and sees)
--source=view1	tells to extract data from viewfinderpanoramas.org view1 means 1 arc second resolution Note: for some countries, this resolution is not available Must use "3 arc seconds" instead --source=view3
--max-nodes-per-tile=0 --max-nodes-per-way=0	mandatory to have only one output file
--pbf	output is a .pbf file
-o PREFIX	the output file will be PREFIXlon-xxx-lat-yyy-view1.osm.pbf

Installation : the old tool

The original tool (the one described in Internet tutorials for creating contours) was Phyghtmap
Unfortunately, it seems no more maintained, and is incompatible with Ubtuntu versions later than Ubuntu:focal
(incompatible evolution of the 'matplotlib' library)

This tool does not exist in standard Ubuntu/Debian repositories.

A .deb package has been downloaded from
http://katze.tfiu.de/projects/phyghtmap/#Download
and copied to directory "docker", so that it can be installed when processing dockerfile

Installation: the new tool

The new tool Pyhgtmap is a clone of Phyghtmap, with plenty of evolutions

It brings some benefits : it uses a more efficient algorithm (Ramer-Douglas-Peucker)

Unfortunately, its installation is much more complex:
- installed by "pip" method, with needs installing lots of stuff in the Docker image
- requires plenty of other small pip packages
- since pip packages are not standard ones, it needs setting up a 'Virtual environment' (we use 'venv') : packages will be installed in a kind of sandbox, that does not pollute standard python
- requires installation of GoogleDrive pip packages, just for allowing access to a specific datasource which is on GoogleDrive

To avoid tons of GoogleDrive packages, the installation/execution steps have been adapted: some tuning has been made, to populate the solution with adequate contents
- Manual actions, incorporated in the project files :
  - manual download of the pip package (without installation) , and uncompress python source files
  - removing of file requiring Google packages
- Dockerfile actions :
  - explicit installation of required dependencies (excepted Google)
  - execution through a bash script (pyhgtmap.sh) which directly invokes entry point (main.py) from the 'virtual environment'

6.3 Loading contours with osm2pgsql.org

If you modify pyhgtmap parameters to produce a pure XML output, you will see that its contents is quite simple: a huge list of <node> elements and several <way> elements referring to those <node>

The altitude information is located in the <way> element , using tagname "ele"

```
<way id="10000003" version="1">
        <nd ref="10000041"/>
        <nd ref="10000042"/>
        <nd ref="10000043"/>
        <nd ref="10000044"/>
        <nd ref="10000041"/>
        <tag k="ele" v="860"/>
        <tag k="contour" v="elevation"/>
        <tag k="contour_ext" v="elevation_minor"/>
</way>
```

osm2pgsql will need a specific reduced .style file :
- saying that a column named "ele" must be created, based on tagname "ele"
- forcing the creation of another column "elegroup" which will be used for optimizations

contours.style

```
# OsmType    Tag         DataType    Flags
node,way     ele         int4         linear
way          elegroup    int4        linear    # to force creation of specific column
```

We will import contours into a dedicated database "contours"
osm2pgsql --slim --drop -d contours --cache 1000 --style ./contours.style ./contoursfile.osm.pbf

***Note**: why mention "node,way" rather than just "way" for the tag "ele" which appears only for <way> nodes ? Just because otherwise, there will an error message from osm2pgsql ...*

Notes:
Due to "-s 10" parameter, the .osm.pbf contains only lines whose colun "ele" is multiple of 10

6.4 Optimizations

We want to avoid patterns like "WHERE ele IS NOT NULL AND MOD(ele::numeric, 100) = 0" in SQL requests executed for rendering. (*The syntax "ele::numeric" means converting the text value into a numeric value*)

So column "elegroup" is computed from "ele" to contain values (10, 20, 50 , 100)
It will also allow to create efficient index

For this purpose, a SQL script "zindexcontours.sql" is executed after osm2pgsql import

6.5 Rendering contours

6.5.1 project.mml

Three layers are declared : contours100 , contours50 , contours10

```
  _parts:

# reusable block, defining specific database for contours
  osm2pgsqlcontours: &osm2pgsqlcontours
    type: "postgis"
    dbname: "contours"
    key_field: ""
    geometry_field: "way"
    extent: "-20037508,-20037508,20037508,20037508"
```

```
- id: contours100
  class: contours
  <<: *extents
  Datasource:
    <<: *osm2pgsqlcontours
    table: |-
      (
        SELECT
          way, ele
        FROM planet_osm_line
        WHERE elegroup =100
      ) AS data
  geometry: line
  properties:
    minzoom: 11
    status: on
```

Note the inclusion of block <<: *osm2pgsqlcontours telling the usage of database "contours" and the fact that the GEOMETRY column is "way"

Column "ele" is included in the output of SQL request, because it will be used by .mss stylesheet to draw the altitude value

The layer is associated to class "contours" that will be used in .mss stylesheet

6.5.2 .mms stylesheet

```
.contours {
  line-color: @contours-color;

  /* 100 m */
  #contours100 {
    [zoom >= 11] { line-width: 0.2; }
    [zoom >= 12] { line-width: 0.4; }
    [zoom >= 13] { line-width: 0.5; }
    [zoom >= 14] { line-width: 1;  }
    [zoom >= 16] { line-width: 1; }
  }

  #contours100{

  [zoom >= 15] {
      text-face-name: @standard-font;
      text-size: @contours-larger-font-size;
      text-fill: @contours-fill;
      text-halo-radius: 1;
      text-halo-fill: @contours-halo-fill;
      text-placement: line;
      text-label-position-tolerance: @contours-position-tolerance;
      text-spacing: @contours-spacing;
      text-min-path-length: @contours-min-path-length;
      text-max-char-angle-delta: @contours-max-char-angle-delta;
      text-name: "[ele]";
      }

  }
```

line-color is defined for all layers of class "countours"

Here, the altitude for the contour100 lines is shown only for zoom >=15

Note the syntax `text-name: "[ele]";` which tells to get value from column "ele" of the SQL request

7 Drawing hillshade

Hillshade is the fact of drawing a semi-transparent layer, giving a 3D impression by showing the shadow of mountains and hills.

This uses a famous collection of tools: "gdal"

We directly reuses the DEM data downloaded during the previous step

The trouble is that pyhgtmap downloads .zip files for srtm1 data, which contain much more .hgt files that necessary for the desired perimeter

There are 3 steps

- Build the list of .hgt files necessary to cover the area defined by the polygon file myfiles/xxx.poly. This file "demdata/xxx.poly.txt" is produced by running a small script dem/hgtlist.py
- Use gdalwarp to merge files
- Use gdaldem to produce hillshade

Important notice!
Those tools use compression. Do not choose compression algorithm which alter data (like JPEG).
Output file would be largely smaller, but the image produced by the tile rendering software would contain many ugly pixels (despite the fact that opening the file itself in the file explorer, shows no trouble)

7.1 gdalwarp

This tool will do 3 actions

- get all the .hgt files declared in xxx.poly.txt
  (using the option --optfile)
- merge them into a .tif file (GeoTIFF format)
- make a model conversion between .hgt files (using EPSG:4326 model) to EPSG:3857 used by openstreetmap

An important parameter is -tr mmm mmm , which means that 1 point in the output .tif file corresponds to mmm meters in real world

```
gdalwarp -co BIGTIFF=YES -co TILED=YES -co PREDICTOR=2 -co COMPRESS=DEFLATE
-t_srs EPSG:3857 -r bilinear -rcs -order 3 -wo SAMPLE_STEPS=100 -tr 30 30
--optfile xxx.poly.txt warp-30.tif
```

gdalwarp options are quite cryptic ... and refer to deep knowledge of image transformation.

Here a few explainations:

-co BIGTIFF=YES	produces BigTIFF output files (evolution of the TIFF format to support files larger than 4 GB
-co TILED=YES	the output file is internally tiled, to optimize access to subparts of the file
-co PREDICTOR=2	tells the Compression algorithm to store the difference between neighbouring cell values, rather than the values themselves. This method is more efficient when values do not abruptly change between adjacent cells (maybe 10% gain)
-co COMPRESS=DEFLATE	apply DEFLATE compression algorithm for Alps area, seems a little better than LZW
-t_srs EPSG:3857	Tells what is the Spatial Reference System to be used by the output file (target) The SRS of the input files is deduced from their metadata The appropriate conversion will be done The value here defines WebMercator projection used by openstreetmap and google, where earth is considered as a pure sphere
-r bilinear	use "bilinear resampling" the output value is based on a linear interpolation, using the 4 adjacent input cells
-rcs -order 3	specific algorithm said to be efficient
-wo SAMPLE_STEPS=100	used to improve quality
-tr 30 30	Target Resolution 1 point in output file corresponds to 30meters in real world when this value is small, the output file size and computation time increase For showing hillshade at zoom level 15 or 16, using "30" is quite necessary, otherwise the mountains are "smoothed" with resolution 100 , filesize is 10 times smaller .... with resolution 30 filesize is 1GB for whole France
-optfile xxx.poly.txt	tells that xxx.poly.txt contains the list of input files
warp-30.tif	output file for resolution 30

7.2 gdaldem

The "hillshade" option of this tool will produce the final hillshade .tif image (8 bits raster)
It simulates the shadow created by the sun at 45°

```
gdaldem hillshade -z 3  -compute_edges  warp-30.tif    hillshade-30.tif
```

option -z is the "scale factor" : tells how much vertical distance is exagerated compared to horizontal distance
-z 3 looks fine .
-z 10 will create very dark shadow.

You may try option `-multidirectional` (simulates light coming from different directions) which limits darkness

7.3 project.mml

It seems that resolution and scale factor should have different values depending on zoom level

low zoom	since a large area is shown, use 500m resolution to minimize filesize use -z 10 so that mountains are well visible
high zoom	the hillshade should not be too dark (it would trouble the drawing of all the other detailed information needed) use 30m resolution , but limit scale factor to -z 3

```
- id: hillshade-low
  <<: *extents
  Datasource:
    type: gdal
    file: demdata/hillshade-500.tif
    format: tiff
  geometry: raster
  properties:
    minzoom: 4
    maxzoom: 9

- id: hillshade-high
  <<: *extents
  Datasource:
    type: gdal
    file: demdata/hillshade-30.tif
    format: tiff
  geometry: raster
  properties:
    minzoom: 10
```

explaination:

type: gdal	tells to use the "gdal" plugin of Mapnik rendering tool
file:	tells the name of the .tif file
geometry: raster	tells it is a raster file

7.4 .mss stylesheet

base.mss contains this layer, positionned after landuse and before roads

Each pixel from the hillshade raster image is combined with the pixel produced by the lower layers (those already rendered low layers are named: "base layer")

```
#hillshade-low{
  raster-scaling: bilinear;
  raster-comp-op: multiply;
  raster-opacity: 0.25;
}

#hillshade-high{
  raster-scaling: bilinear;
  raster-comp-op: multiply;
  raster-opacity: 0.25;
}
```

raster-scaling: bilinear;	use bilinear interpolation
raster-comp-op: multiply	tells the mathematical operation used to combine pixels from hillshade and base layer Here the base layer value is multiplied by hillshade pixel value, from 0=black to 1=white . It darkens the base layer
raster-opacity: 0.25;	0 will show no shadow 1 will make final image very dark

7.5 More details:

https://docs.gimp.org/en/gimp-concepts-layer-modes.html
(math description of different raster-comp-op operations . not all are supported by gdal)

https://github.com/mapbox/mapbox-studio-classic/blob/mb-pages/docs/studio-classic-manual/05-classic-manual-image-compositing.md
(list of operations that seem supported)

https://gisgeography.com/bilinear-interpolation-resampling/
(about bilinear interpolation)

https://gdal.org/en/latest/programs/gdaldem.html
https://gdal.org/en/latest/programs/gdalwarp.html
(gdalwarp / gdaldem options)

8 A look at Cyclosm/Osmhike implementation

The initial structure of project.mml and stylesheets, coming from Cyclosm (itself based on openstreetmap) is extremely complex. Especially for roads...

The initial intention was to keep it mostly unchanged (avoid having to analyse this complexity) .
BUT:

- Track/bridleway/footway have been unified . track/path were changed
- Drawing of labels (roads / towns / mountain places) were changed
- And finaly, a major simplification was made in the implementation

8.1 A major simplification

The core of the problem is the need to change the size of roads, depending on the zoom level.

This was initialy done by using a huge set of constants like:

@rdz18_primary	*line size for zoom 18, for roads of type highway='primary'*

The result is an incredible nightmare, where the line width for each feature (road width, casing width, left border, right border, oneway marker ...) is repeated

- for each kind of road
  - for each zoom value in the range [11..18]

I guess that the final file roads.mss has probably been made by automatic generation (because the same patterns are repeated with a strange regularity) . But the result is something extremely complex and difficult to modify...

The solution

All the many line-width definitions are replaced by only 3 lines:

- line-width: [roadsize];
- line-width: [roadcase];
- line-width: [sizecycle];

The values [roadsize] , [roadcase] , [sizecycle] are computed by the SQL request, using patterns like:

```
SELECT
        ....
        zfact_roadsize(   highway ,    !scale_denominator!   ) as roadsize
```

So, there are 4 personnal built in SQL functions:

zfact_roadsize(highway ,!scale_denominator!)	*returns the base line size of road/track/path*
zfact_roadcase(highway ,!scale_denominator!)	*returns the line size for the extra casing borders*
zfact_sizecycle(highway ,!scale_denominator!)	*specific line-width for cycleways*
zfact_watersize(waterway ,!scale_denominator!)	*specific line-width for waterways (canal, stream ...)*

The first argument is the road type . The second argument contains the magic token !scale_denominator!, which is used to guess the zoom. *(There is no exact computation from scale_denominator to zoom, because it would be too much time consuming, but the function compares scale_denominator to key values)*

Using csv file to define values...

This is an original way for defining values, which offers a better global view.
And since this stuff runs on Linux, having LibreOffice to edit .csv files is not a problem ...

File zfact.csv contains the line-width values for different zooms, with patterns like:

skip	function	key	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	roadsize	motorway								2				3			4	6	10	16	23		
	roadsize	primary								1				3				5	8	14	20		
	sizecycle	*												0.4	0.7	1	2			3	4		

This table is to read from left to right . We must put values only for zooms when there is a change in linesize

- concerning 'roadsize' when road is of type 'motorway'
  - if zoom >= 18 , then roadsize=23
  - if zoom >= 17 , then roadsize=16
  - ....
  - if zoom >= 7 , then roadsize=2

- concerning 'sizecycle' whatever the type of road (* means 'any value')
  - if zoom >= 18 , then sizecycle=4
  - ....

Note that you can use either , or . for decimal separator...

The column 'skip' , if not empty, is used to comment out a line

A specific script zfact.py transforms this zfact.csv to zfact.sql which defines the SQL functions
The script is inserted into the database, before running tile tirex/kosmtik

Caution

When making changes to zfact.csv, the Tirex/Kosmtik container must be stopped and restated, so that zfact.sql is recomputed and updated inside the database. Just using the feature 'reload' of Tirex/Kosmtik will not be enough ...

For which benefit ?

More than 3000 lines of code suppressed !

8.2 Roads

This manages railway, different kinds or roads , and track , path
Implemented with 3 layers for 3 zoom level ranges : roads_low , roads_med , roads_high

For roads_high , data are extracted from a specific database view "cyclosm_ways" (see views.sql)
It creates new columns:

type	computed from 'highway' tag , with some minor adaptations (combine motorway/trunk ...)
surface_type	analyse 3 tags : smoothness , tracktype , surface to define the compatibility with biking the result is 4 possible values : unknown,road,cyclocross,mtb (MounTainBike) It will be highly used to define the linestyle used for track/path For path, style has been modified, so that in case of no information ('unknown'), we take the worst possible case ('mtb') Note: it is possible to find a path in mountain, having surface_type=road !
alpine	has been added by Osmhike, to distinguish difficult mountain paths 5 values , based on tag 'sac_scale' : • no (not mountain) • low ('mountain_hiking') • medium ('demanding_mountain_hiking') • high ('alpine_hiking') • veryhigh ('demanding_alpine_hiking','difficult_alpine_hiking')
can_bicycle	uses tag 'bicycle' ,and some other tags *(`bicycle` specifies legal restriction for cyclists when used on roads and path)* most interesting resulting values are: 'no' : forbiden to bicycle 'designated' : bikes should go there NULL This column is highly used to differentiate the rendering of roads/track/path
maxspeed_kmh	
motor_vehicle	if 'no' indicates that the road/street is not allowed for motor vehicles
cyclestreet	
oneway	
cycleway_left_render cycleway_right_render	information about the level of security of the cycleway: sidewalk/lane/busway/shared_track
cycleway_left_oneway cycleway_right_oneway oneway_bicycle	tells if oneway or not
has_ramp	is there a ramp
service	standard osm tag, with some adaptations
mtb_scale	based on mtb:scale (mountainbike difficulty)
mtb_scale_imba	based on mtb:scale:imba (another norm for mountainbike difficulty)
tunnel	merging of some tags indicating a tunnel

Stylesheets : road-colors.mss roads.mss

	The layer #roads_high is drawn several times, using :: syntax
#roads_high::outline	defines the external border of the road NOT USED for track/path line-color use constants like @primary-case; the width is computed to add the space for the 2 borders
#roads_high::inline	defines the internal contents of the road also APPLIES to track/path line-color use constants like @primary-fill;
#roads_high::cycleway_left #roads_high::cycleway_right #roads_high::steps_ramp_left #roads_high::outline_left #roads_high::outline_right #roads_high::path_outline_right	draw specific border to indicate the existence of cycleway or bicycle ramp

also note that some stuff is made when a road encounters a tunnel : color is modified

8.3 Managing tunnels / bridges

Each kind of road has its specific color... The goal was to avoid having several line-color definitions (one for each kind of road entering a tunnel)

The solution for roads inside a tunnel:

- the layers "roads_high" , "tunnel" , "bridge" have been merged:
  we use only "roads_high" , with 2 attributes [tunnel!='no'] [bridge!='no'] to define tunnels and bridges
- a white background is drawn
- a dashed casing is added
- a small line-opacity is added, so that the roads is drawn with its normal color, but very light.

8.4 Labels

8.4.1 Roads / Routes names

A road or a bicycle route has 2 names, defined by 2 different tags

- "ref" : short name like A50 , GTV
- "label" : long name like "Autoroute du soleil" , "Tour du Mont-Blanc"

REF will be drawn using a SHIELD

LABEL will be drawn by just printing text, with adequate color

- for roads : inside the road borders
- for biking routes : outside the road/track/path borders

8.4.2 Town labels

Uses 3 layers

Layer	Description	placenames.mss patterns
capital-names	capitals	#capital-names
placenames-medium	city/town	#placenames-medium::high-importance #placenames-medium::medium-importance #placenames-medium::low-importance
placenames-small	village and smaller sub parts of a town	#placenames-small::village #placenames-small::suburb .....

City/Town/Village are rendered:

- with a SHIELD at lower zooms (a nice circle icon , like in road maps)
- with a TEXT at higher zoom

The zoom value at which we move from SHIELD to TEXT rendering is smaller for towns than villages

Inside the SQL rquest, a column "score" is based on this population, but is augmented if the town is a regional capital. The SQL request orders places by decreasing score, and computes a column 'row_number' , so that the .mss style can decide how much of them will be printed (avoids too many names on the map)

8.4.3 Mountain points of interest

This is for

- natural=(peak,saddle,cave_entrance ,volcano)
- tourism=(wilderness_hut , alpine_hut)

Suprisingly, inside osm data,some of them are defined as POINT, some others as POLYGON

Initially, they were part of the great familly of "amenities" , which are rendered by 2 objects:

- a MARKER for the icon
- a separated TEXT for the label

Finally, mountain TEXT object have been separated from amenities objects, to be sure that they are not hidden by overlapping troubles

- text-allow-overlap is enabled
  (of course this may cause overlap when there is a line of mountain peaks, but in this case, overlapping is better than showing nothing)

Layer	Description	.mss patterns
#amenities-points, #amenities-poly	MARKER object a dedicated column 'feature' is computed for different cases	amenities.mss filter on [feature]
#moutain-point-text #moutain-poly-text	TEXT object	labels.mss filter on [natural] and [tourism]

9 Legend building

Manually design the Legend ? Certainly not !

Building the Legend will use a giant trick:

- imagine a fake country, typically at lon=0,lat=0 where there is nothing
- populate it with polygons and lines that represent different types of land usage, roads ...
- generate a .osm file for this fake country, import it to database
- just let Komstik paint it with the right colors and patterns
- download the tiles corresponding to this fake country, then combine them to a final legend.jpg file

Of course, making the .osm file should also be automatic !
For this goal, we write a very simple legend.txt file that describe the different objects to put in the Legend (with the adequate tags)
Then a nice python script legend/makelegend.py will transform it into legend.osm

9.1 Legend.txt

It describes the objects to put in the Legend

Each line look like:
type;tag=value,tag=value;description

type	'poly' or 'line' depending on the kind of drawing to generate (a rectangle or a line)
tag=value	different OSM standard tags, whose value determine the nature of the object
label=description	description is the legend text, printed at the right of the object

Objects will be drawn on a same line, from left to right, until the pattern 'block;' tells to begin a new line

It is not so easy to find out the right attributes to use for defining an object.
Exploring project.mml and views.sql will be of a great help
*Pay attention that project.mml often mentions attributes which **are not** OSM standard attributes found from .osm data file, but new column names created by views.sql. Only the OSM standard attributes must be used ...*

9.2 Operating mode

CAUTION

The following actions are useful only if you are not satisfied with the pre-existing legend/legend.jpg
They will replace the database contents, so it may be a good idea to first save it

```
sudo mv database database-save
```

Step1:

Create the fake country legend.osm from legend.txt , and import it to database

```
sudo docker-compose run import legend1
```

Step2:

Inside project.mml , modify the item "status_contours: &status_contours" , to exclude contours generation
(otherwise map generation will fail due to missing "contours" database)

Restart Kosmtik (or use the "reload" Kosmtik button), so that he reloads map from database
You may tell him to show the legend with : http://localhost:8888/style/#15/0/0

Step3:

Once Tirex/Kosmtik server is ready to serve the legend tiles, build legend.jpg

```
sudo docker-compose run import legend2
```

This will call again makelegend.py to find out which tiles correspond to the legend latitudes/longitudes , download those tiles, then merge then into a simple .jpg file

10 Hints

10.1 Merging several .osm.pbf files

osm2pgsql cannot import several .pbf files

Even if the command accepts several .pbf files as arguments, this will probably fail with errors like "duplicate index"

The solution is to previously merge the .pbf files using magic tool "osmconvert" from package "osmctools"
But this tool can only merge .o5m files, so conversion is needed

Operating mode:

```
# convert each .pbf file to .osm
osmconvert  file1.osm.pbf   -o=file1.osm.o5m
osmconvert  file2.osm.pbf   -o=file2.osm.o5m

# merge files and convert result to .pbf
osmconvert file1.osm.o5m  file2.osm.o5m -o=mergedfile.osm.pbf
```

A small script "scripts/merge.sh" has been written to automate this
It must be run inside a "import" container, (so that necessary packages are available)

```
# run a shell inside a new temporary "import" container
docker-compose run import bash

# .pbf files are supposed to be located in "myfiles" directory
cd scripts
bash merge.sh  mergedfile.osm.pbf      file1.osm.pbf  file2.osm.pbf
```

Note

- Another tool exists: "osmium merge" , but it seems that its result won't work ("duplicate index" error)
- it is highly recommended to use .pbf files generated at the same date, so that they are consistent

10.2 Working with different scenarios

Let's imagine that you have imported all France, but want to make tests and style adaptations on the small Andorra dataset, without destroying previous work...

The best solution is just to rename directory "database" , and let db service create a new one
(stop the db service, before renaming directory)

Another solution is to use different database names

docker-startup.sh	modify variables DB_OSM DB_CONTOURS

project.mml	modify "dbname" in those sections:
	
	osm2pgsql: &osm2pgsql
	  dbname: "osm"
	
	osm2pgsqlcontours: &osm2pgsqlcontours
	  dbname: "contours"

11 Docker implementation

11.1 Introduction

The implementation is a little more complex that the original one from Cyclosm
There are 2 goals:

- allow to create a dedicated volume binding to store database (facilitate backup)
- minimize the size of the "build context" (see docker mini tutorial for details)

The implementation uses a mixed philosophy:

- During image build phase (RUN / COPY dockerfile commands) files from files/docker are executed or copied to the image filesystem
- During image run phase (ENTRYPOINT / CMD dockerfile commands) project subdirectory "files" is mapped to the container /osmhike directory, which allows to directly use files from /osmhike (even for the startup script defined by ENTRYPOINT)

***Note**:*
*Remind that dockerfile RUN/COPY commands **cannot** use files out of the "build context" which is set to directory "files/docker" .*

11.2 Docker Structure

This implementation is based on 3 docker "services" , described by docker-compose.yml

Service	Image name	Container name	Description	Startup script
db	db	c-db	permanently running database server this is the official docker image "mdillon/postgis" from docker repositories	the internal startup script
import	import	c-import	temporary running container, used to import osm data and elevation data to database based on Debian (bookworm version)	docker/docker-startup.sh import
kosmtik	kosmtik	c-kosmtik	tiles rendering web server must be run while you need the tile server based on Debian (bookworm version)	docker/docker-startup.sh kosmtik
tirex	tirex	c-tirex	Alternate tiles rendering web server must be run while you need the tile server based on Debian (bookworm version)	tirex/tirex.sh

Whatever container c-import is running or not, it is possible to execute a **temporary new** container from image "import", to execute specific import actions
A subtle configuration of the dockerfile sections "ENTRYPOINT","CMD" has been made for that

docker-compose run import contours	import contours into specific database
docker-compose run import hillshade	build a raster file used for rendering hillshade
docker-compose run import bash	run shell commands in the container environment (with adequate packages installed) *if you want to install packages, you must run "apt update" before, otherwise they may be not found*

Warning:
Do **not** use "docker run" command, instead of docker-compose, because the network/volumes environment would be missing, causing strange error messages

Notes:

- Services "import" and "tirex" are declared dependant on "db" .
  Starting them will first check that "db" is built and running. It not, c-db container will be started silently (without being attached to a visible window)

- The database network port (standard port 5432) is **not** exposed to host. So it can be reached only from a running container.
  This prevents conflicts, if the host already runs its own postgres database

- Tirex webserver port 80 is exposed to host as port 8888.
  Komstik webserver port 6789 is exposed to host as port 8888
  So it can be reached by http://localhost:8888 or http://host-ip-adress:8888 from any machine on the local network having network access to host

- Environment variable DBHOST is set to indicate who is the database server

Volume binding

- The host local subdirectory ./files is mapped to directory /osmhike inside the containers (import,tirex)

- The host local subdirectory ./database is mapped to database location /var/lib/postgresql/data inside "db" container . This avoids the database to be stored in a mysterious place inside the Docker infrastructure, and makes backups easier.

caution!
When the binding occurs, the container's directory is replaced by the host directory. This means that if the container's directory had some contents, it is completely lost !

USERID

- import,tirex containers run with USERID=1000
  I expect that this userid is the owner of host local project directory (or has full r/w access to it)
- db container runs with root
  Otherwise there are problems for modifying database directory

Which OS version ?

The import/kosmtic images coming from original Cyclosm project were all based on "Ubuntu:bionic"
I made the adaptations to use the latest LTS ("LongTimeSupport") version Ubuntu:noble
Finally, I moved to Debian:bookworm, which is much more stable (incompatibilities suddenly raised with Ubuntu:bionic)

For "import" image, I found troubles if using a version more recent than "focal"

- image building is no more automatic, since the installation of some packages waits for user-input relative to timezones
  => use ENV DEBIAN_FRONTEND=noninteractive
- with latest versions, phyghtmap fails because of an incompatible evolution of the 'matplotlib' library
  => use pyhgtmap instead

For "kosmtik" image, the version got by "npm install" did not work. I had to directly use more recent source from Github repository (but not the latest one, due to sudden incompatibility)

About Tirex

See dedicated chapter

About Kosmtik

It contains 4 components

- A lightweight web server to serve tiles
- Web page for navigating into tiles
- Mapnik renderer to build tiles
- Carto to transform project.mml into .xml files required by Mapnik

Kosmtik has a nice feature:
If the Datasource parameter of a Layer is a url to a zipped shapefile, rather than a SQL request, then he will do the following actions:

- Download the zipped shapefile from Internet
- Uncompress it to the "data" subdirectory of **the directory containing project.mml**
- Patch project.mml , replacing the url by the path to uncompressed shapefile

Notes:

- Do not search for the project.xml file !
  Kosmtik makes a in-memory compilation of project.mml, without writing the result to filesystem

- Kosmtik creates a "tmp" subdirectory **in the working directory where it is executed from**
  It is used as a cache for tiles

Kosmtik url
If the full path of project.mml is /one/two/three/project.mml , then

- the map url showed by Kosmtik is http://localhost:8888/three/#ZOOM/LAT/LON
- http://localhost:8888 will always give access to the world map, at low zoom

Components to install

Docker is the only thing to install on your computer.

All other components will be installed inside the containers, using either official Linux repositories, Python package manager (pip) or Nodejs component manager (npm) .
(There is no need to make complex manipulations on repositories and GPG keys)

Environment file

.env allows to define environment variables

- technical stuff , that you will not change unless good reason
- project parameters, which are initialized to cover the small and quick example of Andorra.
  those files are located inside "files/myfiles" directory

AREAOSM	.osm.pbf file containing osm data
AREAPOLY	.poly polygon file to draw contours and hillshade It may cover a much larger region than AREAOSM
AREAPOLYSOURCE	tells if we use if we use "1 arc second resolution" or "3 arc second resolution" elevation data from viewfinderpanoramas.org (1 arc second is not available for all countries)

Access to database

The "db" service simulates a server named "db" on the local Docker network
(this is the name of the service in docker-compose.yml , not the name of the container)

Both "import" and "tirex" ,"kosmtik" services define environment variables for proper access to database
PGHOST=db
PGUSER=postgres

You can run psql commands:

from currently running c-db container	sudo docker exec -it c-db bash
from a temporay new import container	sudo docker-compose run import bash

Indexes and performance

Most sql request are like "SELECT .... WHERE attribute=value"

This means that creating database indexes on those attributes will produce a major performance improvement
(not necessary to scan all objects in database to find out the right ones)
This is particularly important for low zoom, where the area to draw contains a huge amount of objects

This is achieved by executing zindex.sql at the end of import phase

Inject parameters into postgres configuration file

When building "db" image, script postgres-conf.sh is inserted in directory /docker-entrypoint-initdb.d , so that the startup script automatically runs it

- **It is executed only once**, at the first run, when database directory is initialized
- it adds an include command in the postgres standard configuration file 'postgresql.conf' , so that our own file files/dbconf/myconf.conf gets included

Host directory files/dbconf/ is mapped to container's directory /dbconf , so that myconf.conf can be directly modified, without rebuilding the image (but modifications need a container stop/up)

This is much simpler than the other method (put environments values in .env , declare them in docker-compose.yml , add a script to store their value in database config file myconf.conf)

11.3 Project tree

docker-compose.yml	master file for docker-compose commands
.env	environment variables defining the project behavior
database	postgres will create database here
files	project files
style	
project.mml , *.mss	project style file and style sheets
zfact.csv	defines linesize for different zooms
symbols	symbols needed by project style
views.sql	creation of views needed by style file
zindex.sql	creation of indexes for performance
zfact.sql	functions returning linesizes, depending on zoom
data	shapefiles downloaded from Internet will be unzipped here
docker	dockerfiles and scripts used by docker build phase
Dockerfile.*	dockerfiles
docker-startup.sh	entrypoint for most import/kosmtik services
postgres-conf.sh	script called to modify postgres configuration
pip-install.sh	script to install pip packages needed by pyhgtmap
phyghtmap*.deb	package for obsolete solution phyghtmap
dbconf	contains extra postgres config file
myfiles	place to download osm data files and polygons
dem	scripts for contours and hillshade
contours.style	specific style used by osm2pgsql
zindexcontours.sql	script to create indexes in contours database
pyhgtmap.sh	invocation of pyhgtmap software
pyhgtmap	copy of pyhgtmap software
hgtlist.py	used to build the list of necessary .hgt files
hillshade.sh	build hillshade file for a precision level
demdata	data files produced for contours and hillshade
scripts	various utilities also contains zfact.sh zfact.py used to process zfact.csv
tmp	Kosmtik makes tile cache here
legend	stuff fore creating the legend
legend.txt	describes the topics to put in legend
makelegend.py	script to produce legend
legend.jpg	legend
tirex	files for the Tirex alternate solution
conf	conf files for apache & tirex , copied during docker build phase
html	web page
Dockerfile.tirex	dockerfile
tirex.sh tirex-start.sh	startup scripts
xml-patch.py	download shapefiles & patch project.xml

12 Performance tuning

Here a list of recipes ...

12.1 Inside project.mml

On low zooms, there is the risk that ALL the objects in database would be candidate for drawing, causing a huge extraction of data.

Do not show all layers

Use minzoom/maxzoom to restrict which layers will be shown

Inside a layer, restrict the objects shown

Add a condition on column 'way-area' to exclude objects which are too small

Do not show everything : for example, on low zoom, show only motorways

Use cache-features: true when request is executed several times

This is the case when the .mss stylesheet uses the :: syntax
Without cache-feature , Mapnik will execute the request several times, instead of storing results in memory

When column 'way' is recomputed by the sql request

In this case the standard filter to show only objects matching the BoundingBox, does not work...
You must add yourself the condition: AND way && !bbox!

12.2 Postgres configuration

work_mem

If this value is too small (which is case of the default value), sort operations (ORDER BY) will be done inside a temporary file instead of memory

Caution!
A high value can cause a memory exhaust, since Postgres can try to consume:
work_mem * (number of sort operations inside the request) * (number of connections)

How to adjust it ?

- Log slow requests , and use EXPLAIN ANALYSE to see if sorting is made on ram or disk
- Log the creation of temporary files which are used for data sort (log_temp_files = 0)

effective_cache_size

This tells to Postgres the amount of memory that the operating system is likely to use for disk caching.

Postgres will use this value to take decisions.
If there is a lot of disk cache, it is likely that indexes are already loaded in it, the query planner will try to use them

Recommended value : RAM*0.5 to RAM*0.7 (dedicated database server)
Note: this value includes shared_buffers

shared_buffers

This is the internal Postgres cache. Contains tables data, indexes ...
Recommended value : RAM*0.25

More details ...

https://pganalyze.com/blog/5mins-postgres-work-mem-tuning
https://easyteam.fr/postgresql-tout-savoir-sur-le-shared_buffer/

Logging slow requests

Logging of slow requests is activated by default inside dbconf/myconf.conf
Requests needing more than 5s will be logged in directory database/log
You can rerun them with the sql command EXPLAIN contents-of-sqlrequest; to understand which part takes time.

More details ...

https://explain.depesz.com/s/B9rV
(understanding explain output)

https://kimlaitrinh.me/blog/comprendre-les-scans-de-postgresql/
(understand the concept of 'scan' which is used by EXPLAIN)

Investigate requests performance and impact

There is a magic script: files/scripts/tuning_sql.py originated from osmfr-cartocss

- it takes as input project.mml , a zoom value , and a [lon/lat] point (or Paris as default)
- it defines as BoundingBox, the tile centered on the input point
- it gets all the layers enabled for this zoom value, and the associated sql request
- it replaces "magic tokens" like !bbox! by the appropriate value
- it executes the request, and measures execution time
- an option allows to execute the EXPLAIN sql feature, to get detailed info about performance

The script must be executed from a temporary running container "import" , to have access to database

```
sudo docker-compose run import bash

python3     scripts/tuning_sql.py  --zoom=7 --db=osm --host=$PGHOST --explain=1 --user=$PGUSER style/project.mm
```

12.3 Create additional indexes

Indexes are extremely useful when there is a WHERE condition in a request (on one or several columns) that returns only a subset of existing objects : it avoids reading all objects.

It is highly useful on planet_osm_polygon where many columns have a huge amount of NULL values

The counterpart is that they increase the size of database …
It also increases the workload for inserting data, but in our case, indexes will be created after import, to avoid overhead when populating the database.

Usefull psql commands

\c osm	connect to osm database
\di+	list indexes with their size
\dt+	list tables with their size

Add condition "WHERE xxx IS NOT NULL"

When there are a lot of NULL values, it is a good idea to replace

CREATE INDEX myindex ON sometable (xxx)

by

CREATE INDEX myindex ON sometable (xxx) WHERE xxx IS NOT NULL

It does not really change performance, but reduces the index size by 10 or 20 !

planet_osm_polygon: create index on way_area

A lot of requests on low zooms, use this column to eliminate objects that are too small
Estimated benefit : 2x

13 The Tirex alternative to Kosmtik

Don't like installing a component from a Github repository ?
Too much stuff to install via pip , with many warnings ?
Tons of Nodejs warnings ?
Search for a 2x performance gain ?

An alternative to Kosmtik is possible : Tirex . But it provides much less features

13.1 Different infrastructures

The standard Openstreetmap tileserver infrastructure uses a complex chain of components

- A Postgres database, using the PostGIS extension
- Mapnik for building tiles, based on a .xml style definition
- Carto (also known as CartoCSS) for transforming a yaml format style to the .xml format required by Mapnik
- Apache2 as web server
- Mod_tile (apache component which manages tiles cache, decide whether a tile is obsolete or not, asks a renderer to generate it again)
- Renderd : daemon which gets orders from mod_tile, and uses the Mapnik library to generate tiles
- Leaflet : javascript library, included in a web page, which allows to define a Map as a collection of tiles, and issues the urls to download each of them. Those urls are processed by apache/mod_tile

The Cyclosm infrastructure replaces the chain Carto/Mapnik/Apache/Mod_tile/Renderd/Leaflet by the single component Kosmtik (which silently incorporates Carto/Mapnik/Leaflet)

Tirex replaces Apache/Mod_tile/Renderd : no need to download a Github repository : everything is done by installing packages from official Ubuntu/Debian repository

Tirex is the solution used by Opentopomap
But in our implementation, we keep the yaml format for defining project style. This needs to add Carto in the chain

13.2 Installation

Installation will of course involve several components

Carto	standard package 'node-carto'
Tirex	standard package 'tirex'
Mapnik	silently installed by Tirex
Apache Mod_tile	standard packages: 'apache2' 'libapache2-mod-tile'
Leaflet	standard package 'libjs-leaflet'

We must also write a dedicated html page for showing the map, using Leaflet library
Extra code was added to this page, in order to implement two nice features available in Kosmtik

- show the zoom level
- update the browser url address, when map is zoomed or moved

13.3 Configuration files

The chain will need many configuration files

Component	Configuration file	Origin / description
apache2 mod_tile	/etc/apache2/sites-enabled/osmhike-apache.conf	copied by dockerfile COPY actions contains parameters of the web site serving the map, and associated mod_tile parameters
tirex	/etc/tirex/tirex.conf	copied by dockerfile COPY actions, replacing the file created by the tirex package ... (we just modify some parameters concerning syslog_facility and log_file)
mapnik	/etc/tirex/renderer/mapnik.conf	installed by the tirex package
	/etc/tirex/renderer/mapnik/*.conf	copied by dockerfile COPY actions contains the mapnik configuration for our project. the name of the .conf file is not relevant: what matters is the 'name' parameter inside

Off course, those configuration files must be consistent together, so a few explainations are welcome !

Tileset

Tirex can manage several tilesets in parallel, each of them being associated with:

- A .XML style file
- A specific uri
- A specific tile cache subdirectory
- A name which be used among the different conf files

Each Tileset is associated with 2 mod_tile important parameters, defined in apache configuration osmhike-apache.conf

ModTileTileDir	base directory for tiles cache
AddTileConfig	defines a tileset, according to the following format: AddTileConfig tileset-uri tileset-name This means: tiles will be served by a url like http://localhost/tileset-uri/Z/X/Y.png the cache directory will be ModTileTileDir/tileset-name Our value for tileset-uri is: /tiles/osmhike

tileset-name will be communicated to Tirex, so that he can find the associated configuration.
He will look inside /etc/tirex/renderer/mapnik/*.conf , searching for a .conf file where parameter 'name' matches tileset-name

Note:
tileset-uri does not correspond to a true directory . Mod_tile will automatically catch and process any uri which begins by tileset-uri

/etc/tirex/renderer/mapnik/.conf*

The name of this .conf file is not important. What is important is the "name" parameter it contains

name	must be equal to the tileset-name defined in osmhike-apache.conf
tiledir	must be equal to ModTileTileDir/tileset-name
mapfile	path to the .xml style

Communication socket between mod_tile and tirex

The same value must be found in both:
- osmhike-apache.conf : (ModTileRenderdSocketName)
- tirex.conf : (modtile_socket_name)

Communication socket dir

This directory is used by Tirex to create sockets, used by the different Tirex subprocesses

It is defined in tirex.conf (socket_dir)

Leaflet configuration

The HTML page showing the map, must tell Leaflet that the url to download a tile is:
/tileset-uri/{z}/{x}/{y}.png

We use a relative uri, without http://server:port

Apache configuration

As defined by osmhike-apache.conf:
- apache is internally listening on port=80
- DocumentRoot is set to /osmhike/tirex/html

13.4 Docker implementation

The commands are similar to those used for Kosmtik

```
# build the image
sudo docker-compose build tirex

# start the tirex web server
sudo docker-compose up -V tirex
```

Container name is c-tirex

Its apache internal port:80 is exposed as port:8888 to the host

The web page url is http://localhost:8888

Files come from 4 directories:

files/tirex	Dockerfile and scripts used by ENTRYPOINT
files/tirex/conf	apache / tirex configuration files which are copied to the docker image, during build phase This is not all the apache/tirex config files : some of them are provided by the standard packages
files/tirex/html	Contains the web page, showing the map It is seen by the container as /osmhike/tirex/html (thanks to volume mapping)
files/tirex/html/leaflet	Directory containing leaflet package *but it is a symlink to /usr/share/javascript/leaflet , and is created by startup script*

Automatic processing of shapefiles

Kosmtik had this nice feature when a Layer refers to a shapefile on Internet:

- automatic download of the shapefile, but only if not already done
- unzip
- patch project.mml to replace path to Internet shapefile by a path to local unzipped shapefile

Tirex does **not** offer this feature, so a dedicated script files/tirex/xml-patch.py has been made for it.
The final project.xml file is generated in directory files/style
Downloaded shapefiles are downloaded/unzipped/renamed in directory files/data

Debug

There are so many involved components, that understanding troubles is quite complex !

- from Firefox browser, use web development tools to trace network requests, and see if the tirex container answers to requests, and which of them
- run bash inside c-tirex container (docker exec -it c-tirex bash) and check that the internal apache server accepts connections on port=80
- setup debug mode for Tirex:
  modify files/tirex/tirex-start.sh : uncomment variable DEBUG
  then stop / start container

Beware from cache, when changing project.mml / *.mss !

- clear apache cache
- stop/start Tirex container, without forgetting option -V in docker-compose up
  (to be that sure that new container starts from a fresh filesystem, without cached files)

Access to apache/tirex logs

To facilitate debugging, the apache/tirex log directories of tirex container are mapped to directories files/tirex/zlogapache , files/tirex/zlogtirex of host

Limits

The Leaflet implementation provided in this project is basic, and has a strong timeout bug:
If tile rendering takes too much time, the tile remains gray : must wait a little, then refresh page

More details

https://wiki.openstreetmap.org/wiki/Tirex
(have a look at chapter "logging")

https://github.com/cyclosm/cyclosm-website
(example a more elaborated html page using Leaflet)

14 Docker mini tutorial

Don't know what Docker is ? No panic ! This very short guide gives you the fundamentals

14.1 Basic concepts

Docker is a sotware infrastructure running on your computer (we wil call it the **host**)
It allows to build and run a kind of lightweight virtual machines called **container**

Image
An image is a template that defines a docker virtual machine.
It contains a photo of the disk filesystem + execution parameters

An image is generally is generally composed of

- a base operating system (like Ubuntu version 'bionic'). It is often different from the host operating system
- plenty of software packages which are installed (like Apache, or a personnal software)
- project files which are copied to the image filesystem
- some other configuration stuff (network, volumes , ...)
- the startup command/program to launch when you run the image

Container
When you run an image, it creates a kind of independant virtual machine called **container**
You can run several containers from the same image (eg: if you have an image "Ubuntu + Apache + Mysql + Php" , you can run several containers which will be independant web servers)

Each container has its own copy of the image's filesystem.

Unless specific configuration, containers can't read/write files of the host filesystem

Container status
When you run a container, it keeps running until the command specified by ENTRYPOINT / CMD terminates
Then container moves to state "stopped" ("exited") .

Each container is given a "container id" . But you can also assign a name to it (easier to manipulate)

Building an image
Have a project directory on host computer
Create a file named Dockerfile, containing the instructions to build the image

The main instructions in the Dockerfile are

FROM	the base image, containing the base operating system example: ubuntu:bionic It will be downloaded from the official Docker repository But it can also be a personal image that you have built yourself, which will be got from the internal Docker repository on your computer
RUN	Runs commands used to construct the image's filesystem In most cases, il will be commands for installing packages from standard Linux repositories, using appropriate commands (apt for Ubuntu/Debian , yum for Redhat ...)
COPY	copies files from your host towards the image filesystem
ENTRYPOINT CMD	defines the command to launch when the container is run

You can cascade the creation of images. For example :

- Build image Myapache , based on Ubuntu , having RUN commands to install Apache/php
- Build image MyWebServer, based on Myapache , having COPY commands to copy application files from the project directory

NOTE

- the package installation commands (apt , yum ...) to use in RUN sections, are the ones compatible with the base image operating system (described by FROM) , not the ones from your computer.

Basic Docker commands

```
# Note : all Docker commands must be executed by user root

# build an image, from the Dockerfile in the current directory
cd  myprojectdirectory
docker build -t myimagename  .

# see existing images
docker image list

# run a container, giving it a name
docker run --name mycontainername  myimagename

# show all containers  ( running or exited )
docker ps -a

# show all containers ( full output, not truncated )
docker ps -a --no-trunc

# remove a stopped container ( if the container was not given a name, use the containerID )
docker rm  mycontainername

# remove a container , ( force it to stop if necessary )
docker rm -f  mycontainername

# clean the system : remove all stopped  containers
docker system prune

# remove an image. If a running container uses it, it will be stopped
docker image  rm -f  myimagename

# runs a shell inside  a running  container ( see files , processes ... )
# DO NOT use "docker run mycontainer bash" , because this would create another uninitialized container
docker exec -it  mycontainer bash
```

Network environment

- A virtual docker network is created.
  All the containers, but also the host, have an IP address on this network

- If host has Internet access, containers have Internet access
- If host has some open ports (like a running webserver) , then containers have access to this host port

- A container can freely access open ports on other container (if a "Mysql" container is running, another "Apache" container can access the database)
  But the host cannot easyly do that. The xxx port on the running container must be "EXPOSED" , so that it also becomes port xxx on the host. Requests going to port xxx on host will be redirected to port xxx on container (Some translation can be made, so that port 80 on container becomes port 8080 on host . This allows to run several containers in parallel, each assigned to a different host port)

- This EXPOSE action can be made by 2 ways
  - by the EXPOSE keyword in Dockerfile
  - by an option in "docker run" command

Volume binding

This allows to replace a container directory (like /cyclosm) by a link to a directory in the host (like /home/user/myproject)

This host directory will be shared between host and container: changes made on one side are visible to the other side

This is an option when running the container

14.2 Docker-compose

This nice tool is based on a file: docker-compose.yml , which defines "services"

The description of each "service" contains

- the name of the docker image used by this service
- the name of the container, when we run the image
- the path to Dockerfile used to build the image
- container ports to be "exposed"
- volume bindings to set up when running the container
- the other services which this service depends on
  (eg: a WebServer container depends on a Database container)

Usage:

```
# Build the image associated to a service
docker-compose build   myservice

# Do all the stuff necessary to have myservice running
# *  build the image if necessary
#  * build all other necessary services, if not already done
#  * run all other necessary services, if not already done
#  * run container, with a fresh new filesystem
docker-compose up  -V myservice

# Run a temporary new container associated to a service, with the capability to replace the CMD  section of dockerfile
docker-compose run  myservice somecommand

# stop the container associated to a service, but keeping its filesystem available for the next usage
docker-compose stop myservice
```

Behind its apparent simplicity, docker-compose has a couple of severe, surprising traps:

Caution:
If you stop a container (CTRL/C or docker-compose stop service), the container's filesystem is kept as a Docker "anonymous volume" for future usage
If you start it again:

- with docker-compose up -V service
  the container will be recreated with a fresh new filesystem from its image
- with docker-compose up service
  the container is just restarted: the entrypoint is executed on the previous filesystem contents

This can cause very strange side effects:

- If the container's entrypoint runs a command to append a line to a file, several "docker-compose up service" sequences will cause several lines to be appended to the file
- If container runs a daemon like apache, and there is a file indicating whether apache is running or not, then a brutal termination of the container can cause some strange error messages like "apache is already running"

Parts of the container's filesystem which are mapped to a host directory are not affected by this behavior: they always remain persistent ...

Caution:
Command "docker-compose run" does the volume mappings , but does not expose internal ports to host
(if your image is running apache, it will not be accessible from the host)
To avoid this : add option "--service-ports"

Warning:
If you are using docker-compose, do not use direct commands like "docker run myimage" !
(all the port/volume configuration defined in docker-compose.yml would be missing ... causing surprising error messages)

Environment file

.env file is automatically processed by docker-compose
(must be stored in the same directory than docker-compose.yml)

But it works only for the variables which are declared in the sections "environment:" inside docker-compose.yml

Note that setting a shell environment variable before calling docker-compose, will overwrite .env definitions

14.3 Hints

Disk space optimization

You will note some strange patterns in the RUN section of the Dockerfile

apt update	used before calling apt install necessary to update the list of available packages Otherwise packages may not be found
rm -rf /var/lib/apt/lists/* && apt clean	used after calling apt install used to free more than 100MB of cached data, that would otherwise be included in the image filesystem, making it uselessly too big. Note: due to Docker behavior, this must be repeated after each RUN section that used apt

Beware from build context size !

docker-compose.yml mentions a directory in section "context:"
This is the Docker "Build Context"

It is important that this directory contains only what is necessary for dockerfile commands
The reason is that all its contents will be sent to the Docker daemon (see message "send build context")
So it is not a good idea to have plenty of downloaded stuff or cached files here...

RUN/COPY commands can access host files ONLY inside the BuildContext

During image building, the "Build Context" directory is the only part of the host files which is accessible by the dockerfile (even if there is some "volume mapping")

Volume binding: files not accessible during dockerfile RUN commands

At the moment RUN commands are executed from dockerfile, volume bindings are not yet operationnal : do not expect to use files stored in the directory which is shared between your host and the container ...

If the RUN commands create a file in this shared directory, this file will disappear when the volume binding occurs, at container start