Vector Tiles from OpenStreetMap

Lukas Martinelli, Manuel Roth

Free Vector Tiles from OpenStreetMap data
Student Research Project Thesis, Fall 2015.

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

SUPERVISORS:
Prof. Stefan Keller

UNIVERSITY:
HSR University of Applied Science Rapperswil

DEPARTMENT:
Department of Computer Science

INSTITUTE:
Geometa Lab

LOCATION:
Rapperswil

TIME FRAME:
Fall 2015

ABSTRACT

Creating a custom styled OSM map is one of the most common use cases among cartographers, yet it is very difficult to do so. With the provided free and open source vector tiles it is possible to allow anyone to create their custom OSM maps without having a deep knowledge of vector tiles.

In this thesis a workflow to create vector tiles from *OpenStreetMap* data has been defined and publicly documented, allowing other developers to adapt the workflow and use it for their own vector tiles. The vector tiles are compatible with Mapbox products and therefore provide an easy migration path for companies that want to use free vector tiles but still use existing tools and technologies.

The vector tiles are provided as public download on the project website (http://osm2vector tiles.org) allowing anyone to download the data and build custom maps and solutions on top of it without being bound to an external service.

KEYWORDS:
OpenStreetMap, Mapbox, Vector Tiles, PostGIS

Situation

Web mapping has gone through different technological changes in recent years. After serving static images for an extract of the map, Google introduced raster tile based maps with Google Maps and it soon became the standard for web maps.
Now the major players have shifted to using vector tiles. Vector tiles allow map designers to individually design their own map. The system administrator does not need to manage large infrastructure anymore, as the tile rendering process can be offloaded to the client side. This results in faster maps with a better user experience.

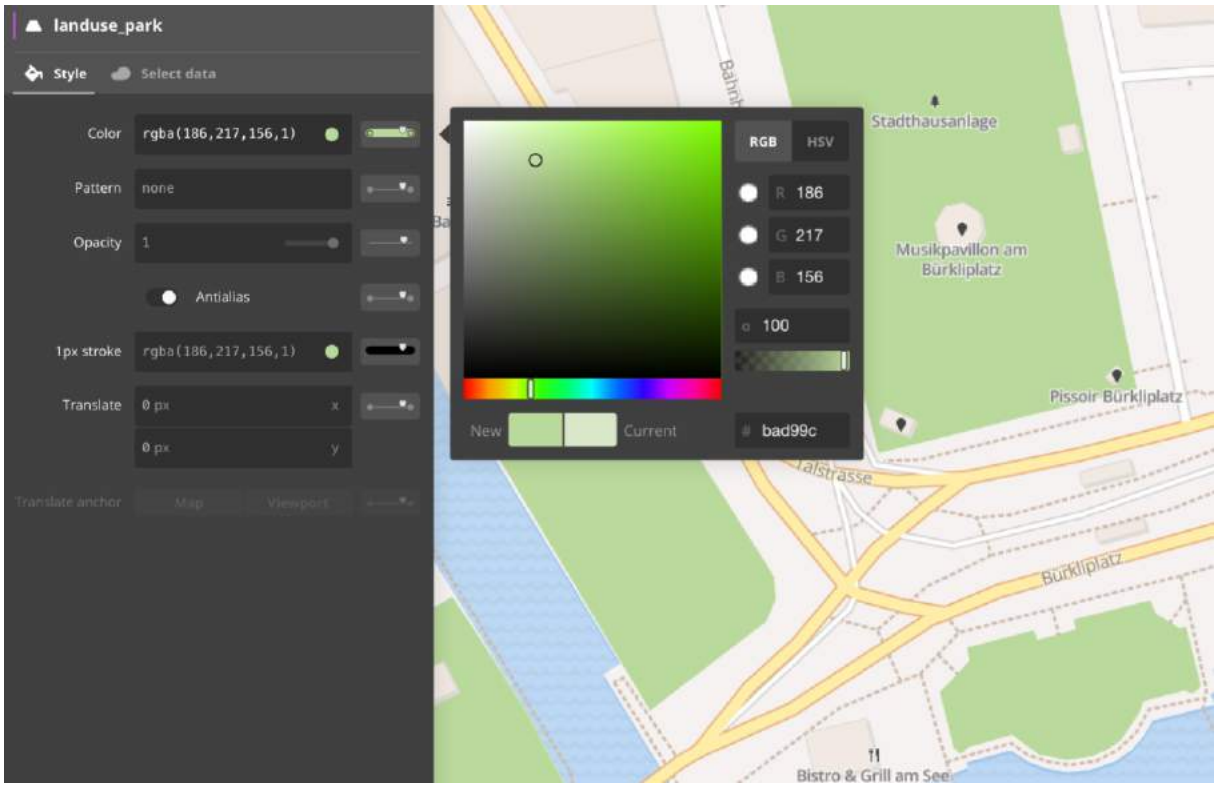

Figure 1: Custom styling of maps in Mapbox Studio

A few existing providers opened the process of creating vector tiles, but still own the data to promote their services. Producing vector tiles requires a good understanding of map technologies and sufficient computing power. This is the reason why vector tiles aren't adopted by the main stream yet.

Approach

The main objective of this project is to create free and open-source vector tiles of Open Street Map data. So that every developer, cartographer or designer can create their custom maps.
An entire workflow for producing vector tiles was defined and a vector tile server to serve the produced vector tiles was implemented. The vector tiles are compatible with the vector tiles of Mapbox Streets, therefore the same visual style provided by Mapbox can be used with our vector tiles. The figure below shows the Mapbox Artistic Woodcut visual style, which can be used with our vector tiles as data source.

Figure 2: Mapbox's Artistic Woodcut visual style

Result

The result of this thesis are vector tiles of Switzerland. They are available for download from the project website (`http://osm2vectortiles.org`). These vector tiles can be served together with a custom or Mapbox visual style in our vector tile server.
As the entire workflow of creating vector tiles is documented and open-source available, other organizations could now use this project to produce vector tiles of their own datasets.

ACKNOWLEDGEMENTS

We want to thank the following people for their support and contributions to the thesis.

Prof Stefan Keller, IFS Institute for Software, for his strong support with regular meetings, contacts in the OSM community and time and effort in checking this thesis.

Dr Petr Pridal, Klokan Technologies GmbH, for his strong support with intermediate technical decisions, project management, regular meetings, donating cloud infrastructure and the CDN infrastructure for hosting the final tiles.

http://www.klokantech.com/

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

OSM	OpenStreetMap, free map
ETL	Extract, Transform and Load
RUP	Rational Unified Process
GIS	Geographic Information System
GDAL	Geospatial Data Abstraction Library
WMS	Web Map Service
DRY	Don't Repeat Yourself
CI	Continuous Integration
CDN	Content Delivery Network

GLOSSARY

Vector Tiles Packets of geographic data, packaged into pre-defined roughly-square shaped "tiles" for transfer over the web

Data Style Description of feature classes such as landuse, water or roads

Visual Style Definition of style rules for a specific schema, which is defined in the data style

Feature Class Group of features with the same geometry type and attributes

Layer Mapbox definition of a feature class

Mapbox Streets Name of Mapbox's vector tile source

MBTiles File format for storing map tiles in a single file

GeoJSON File format for encoding a variety of geographic data structures

Mapnik XML Stylesheet for the mapnik rendering engine

CartoCSS Mapbox propritary cartographic styling language

Mapbox GL Clientside rendering engine

Web GL Javascript API for the graphics library in browsers

Mapbox Studio Classic Client application to design custom maps

OSM Bright Mapbox visual style

Docker Operation system level virtualization on Linux

Kitematic Client application for controlling docker containers

Natural Earth Public map dataset

OSM Planet All OpenStreetMap data in one file

Part I

TECHNICAL REPORT

1 INTRODUCTION

Creating a custom styled OSM map is one of the most common use cases among cartographers yet it is very difficult to do so. With the new emerging technology of vector tiles it is possible to allow anyone to create their custom OSM maps without setting up a database and managing complex infrastructure.

1.1 VISION

Michal Migurski published on March 15, 2013 a blog post[46], in which he described his first attempts to use vector tiles as a source for the Mapnik tile renderer and his vision for vector tiles.
Vector tiles only contain geometries and metadata. The visual style is applied on the fly when the tile is requested, it is separated from the actual vector data. Vector tiles are smaller than raster tiles, this enables high resolution maps, fast map loads and efficient caching.[38]
Our mission is to bring the power of vector tiles to anyone and provide the data free and as Open Source project.

1.2 GOALS

The main goal of this thesis is to allow anyone to create their custom OSM map without managing complex infrastructure. In order to complete this goal several deliverables were defined in the project proposal.

- Create workflow to create vector tiles from *OpenStreetMap* data
- Provide vector tiles for Switzerland
- Provide method to serve the vector tiles together with custom styles as raster tiles
- Optional: Vector tiles for the entire world

DIFFERENTIATION The resulting vector tiles allow creating an alternative base map, which is customizable. The vector tiles are not meant to be queried and do not support custom overlays. Additional map features need to be implemented with the help of other libraries.

2

STATE OF TECHNOLOGY

As of today most web maps are based on raster tiles except a few big providers like Google, Mapbox and Apple. The rest of the industry is now in the transition from raster based maps into vector based maps. For vector based maps the Mapbox Vector Tile Specification[18] is the most dominant Open Source specification of vector tiles.

2.1 CURRENT VECTOR DATA PROVIDERS

While the deployment setup for vector based maps is much simpler than the traditional raster tile setup the most complex part is still the creation of vector data which takes a lot of time and care.

MAPBOX Mapbox provides vector tiles of the entire world as part of its map hosting technologies since 2012 branded as Mapbox Streets[4]. Mapbox also provides the vector data for buyers of their Mapbox Atlas Server[33] starting at $49,000 and distributes quarterly updates for $10,000 per year .

MAPZEN Mapzen provides API access to their public vector tiles[43]. Mapzen states that access and the platform should remain free and Open Source[44]. Mapzen however does not give access to the entire raw data and one is bound to the limitations of the service.

KARTOTHERIAN The Kartotherian[45] project from the Wikimedia Foundation[64] is very similar to this project. The goal is to provide a free Map service that everyone can use for free. The process is documented but the data is still bound to the service and not available as download. The quality of the vector tiles is continuously improved but still lacking very important features. Kartotherian is a great project which the findings of this thesis could be contributed in the future.

GOOGLE AND APPLE Apple started using vector tiles in their Apple Maps product in 2012[80] and Google is using vector tiles since 2013[81] but they are both not accessible for the general public and use a proprietary format.

2.2 CHARACTERISTICS

While the providers open the process they still own the data in order to keep their strategic advantages and promote the use of their products using this data. The open process is a wonderful step in the right direction, yet it requires great understanding of the technology and sufficient computing power to actually execute the workflow of producing vector tiles with worldwide coverage.

2.3 SHORTCOMINGS

The providers already solve the problem of producing the vector tiles and making it accessible. The data however can only be used via their services.
Being able to download world data and use it in a custom server infrastructure, offline and without strings attached is a big advantage and a problem this thesis wants to solve.

3

IMPLEMENTATION CONCEPT

The implementation is divided into the vector tile rendering process and the tile server. The following two sections describe the high level concept of both implementations.

3.1 VECTOR TILE RENDERING

IMPORT In the import phase data sources are imported into a spatial database and mapped to a database schema.

EXPORT In the export phase a data style is applied to the database to produce vector tiles.

DATA STYLE The data style is a description of the vector tile structure and the SQL queries to fetch the data.

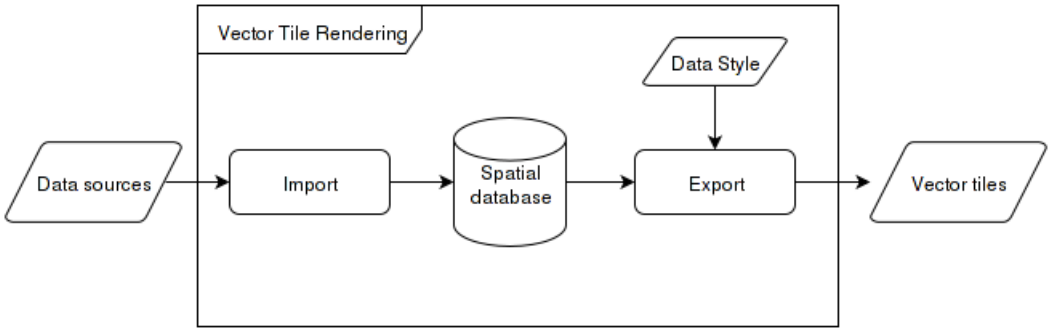

Figure 3: Vector Tile Rendering Process

3.2 TILE SERVER

To display the vector tiles a tile server and a visual style is needed. There are two possibilities, which are described in the following two sections.

VISUAL STYLE The visual style defines how a feature class such as landuse actually gets displayed. In the case of landuse on could define the texture or the color of the border and area.

3.2.1 *Raster Tile Server*

The raster tile server makes it possible to display the vector tiles. It takes a visual style like CartoCSS[13] and the vector tiles as input. The renderer generates raster tiles of these inputs. When a raster tile is rendered, it is served by the webserver. Any GIS Software which supports raster tiles can consume the raster tiles served by the server.

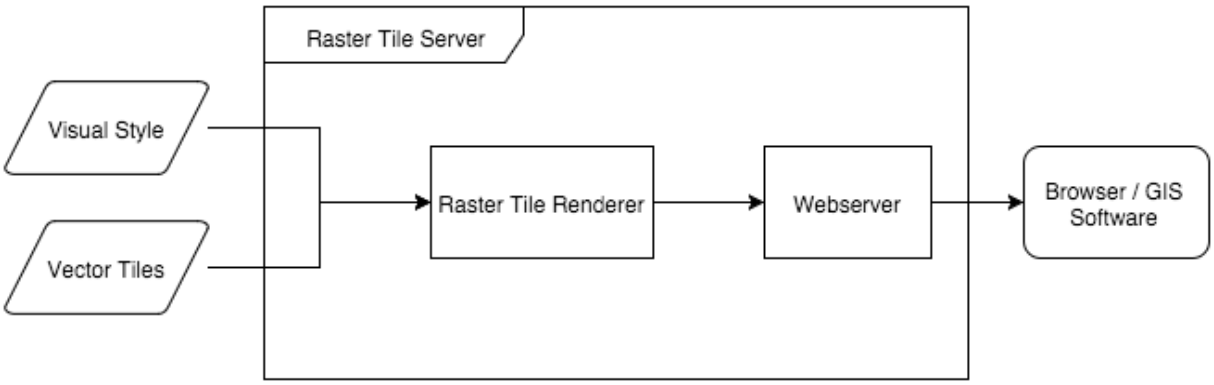

Figure 4: Raster tile server

3.2.2 *Vector Tile Server*

The vector tile server in contrast to the raster tile variant directly serves the vector tiles to the client. The raster tiles are rendered on the client side using visual style. The advantage of this approach is more control over the user experience and that less server computing power is needed.

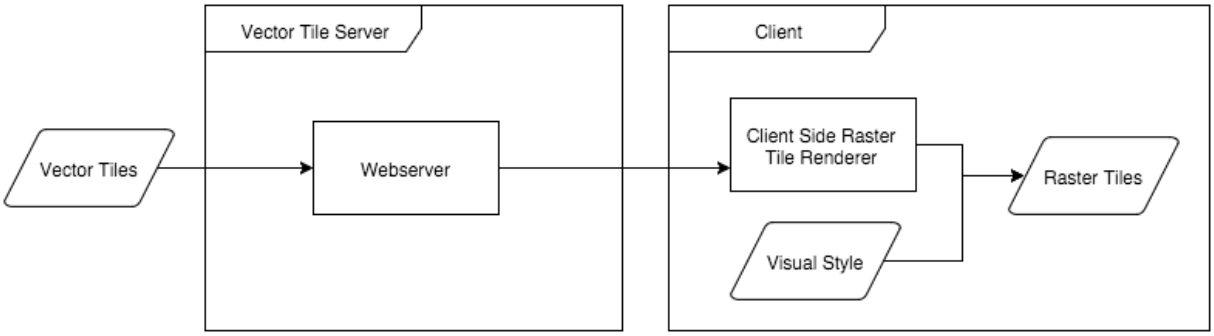

Figure 5: Vector tile server

4

RESULTS AND FUTURE

All objectives which we defined in section Goals have been achieved. The optional goal to provide vector tiles for the entire world has been moved to the bachelor thesis.

4.1 RESULTS

- Docker containers and documentation for the entire workflow of creating vector tiles has been created.
- The raster tile server for serving the vector tiles together with a visual style has been realized.
- The project website with information about how to use the vector tiles is online.

The vector tiles for Switzerland can be downloaded from the project website[1]. These vector tiles can be served together with a visual style in our vector tile server.
A custom visual style can be created with Mapbox Studio Classic[41]. All existing visual styles based of Mapbox Streets are compatible with the produced vector tiles. This allows very easy migration to osm2vectortiles.

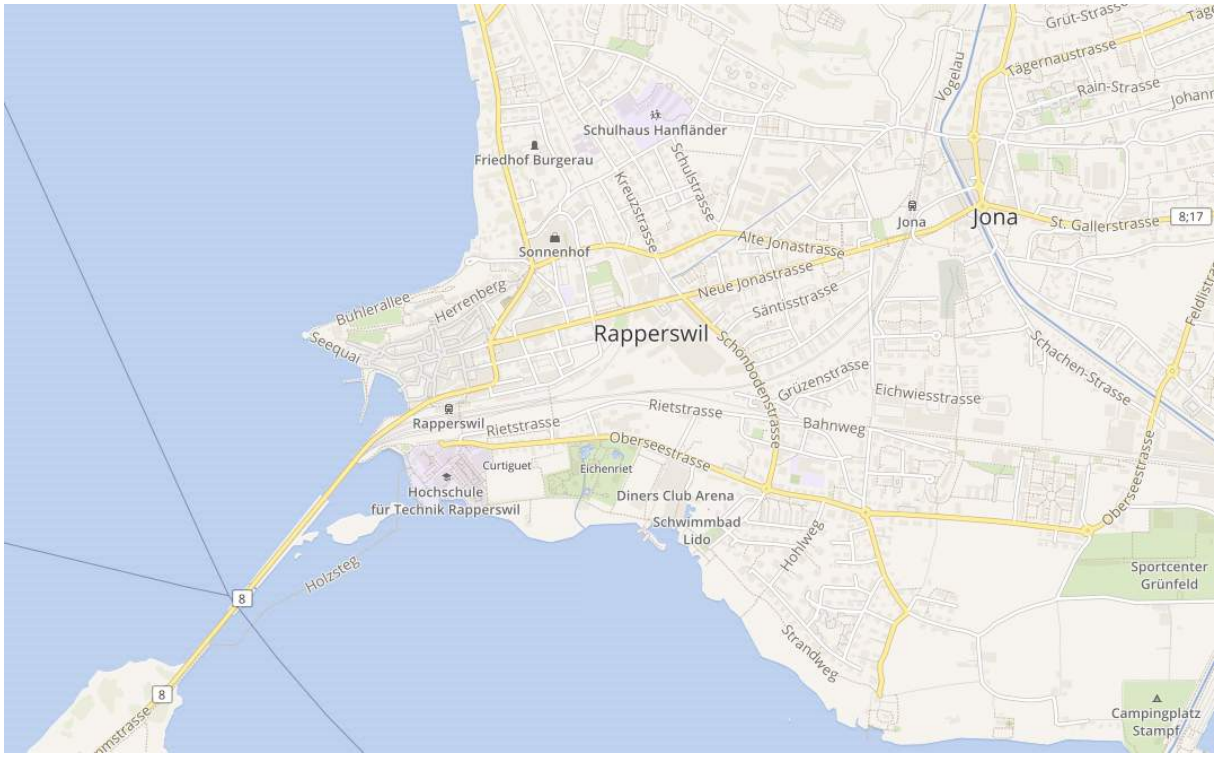

Figure 6: Unmodified OSM Bright visual style using osm2vectortiles

1 http://osm2vectortiles.org

4.2 FUTURE

In a first step the project should be expanded to provide vector tiles for the entire world. The vector tile rendering workflow needs to be scaled out for the entire world. Regular updates for the vector tiles have been requested by the members of the Swiss OSM community and will require identifying and rerendering updated tiles.
The long-term vision of this project is to provide a complete offline map experience, including basic geographic name search. These suggestions will be the project goals for the bachelor thesis. More detailed listing of future features can be found in Chapter 10.

Part II

PROJECT DOCUMENTATION

5 VISION

The vision of our project is described in part 1, section 1.1. This chapter should give a bit of background on where the idea of vector tiles came from.

5.1 HISTORY OF WEBMAPS

Web mapping has gone through different technological changes in recent years. It is important to understand the evolution of web maps to understand why vector tiles are quite a fundamental change in how maps work.

PHASE 1: UNTILED STATIC MAPS In the beginning WMS servers generated static images for the viewport of the map.

PHASE 2: RASTER TILES In 2005 Google introduced Google Maps and XYZ tiles[78] which delivered a idempotent raster image for coordinates specified by a tile index.

PHASE 2.5: RASTER TILES WITH VECTOR OVERLAYS To provide a level of interactivity, tools like Leaflet[32] support rendering vector data like SVG on top of a raster based maps.

PHASE 2.75: RASTER TILES FROM VECTOR TILES For backwards compatibility and faster serving of raster tiles vector tiles where introduced to avoid querying a database.

PHASE 3: VECTOR TILES Vector tiles are delivered directly to the browser and rendered by Web GL based clients.
Improving the use of vector data in web mapping is often shown as the next challenge of web mapping [11, p. 88]

6 REQUIREMENTS SPECIFICATION

This chapter describes the requirements for the vector tiles and the vector tile server.

6.1 USER CHARACTERISTICS

There are three user groups interested in this project:

- **Map Designer**: A technically versed person using Windows or OSX with knowledge of GIS software but not necessarily of it's inner technical workings.
- **System Administrator**: The person which needs to host the published maps.
- **Geographic Institutes or other companies**: Institutions which would like to provide their own geospatial data in vector tile format.

6.2 USER STORIES

SERVERSIDE RENDERING The map designer brings his new map design to the system administrator and tells him to host the map. The system administrator can download the vector tiles of the osm2vectortiles website and serve the map with the help of a vector tile server.

CLIENTSIDE RENDERING The map designer brings his new map design to the system administrator and tells him to host the map. The system administrator does not want to provide a big server that can handle the heavy rendering work. So he decides to use Mapbox GL[42] which offloads the rendering work to the client. In order to use Mapbox GL he needs to statically serve the vector tiles.

OWN VECTOR TILES Institutions which provide geospatial data would like to follow the trend of vector tiles and provide their data in vector tile format. These institutions can use this project for this matter. They can easily modify this project to use it with their data.

6.3 NON FUNCTIONAL REQUIREMENTS

The non functional requirements are the key to success of this project. If the following requirements can be fulfilled, the specified users will be able to benefit form our project.

USABILITY The vector tile server must be usable with Kitematic[30]. Kitematic is an easy to use user interface for docker.

LEARNABILITY Map designers should not have to learn how to use the command line in order to use Docker.

REPEATABLE Generating OSM vector tiles must be possible in a weekly interval because OSM updates regularly.

PERFORMANCE The tileserver must handle 10 concurrent users per second.

COMPATIBILITY The vector tiles must contain all feature sets Mapbox Streets contains. If full compatibility with Mapbox Streets[36] can be guaranteed all Mapbox visual styles can be used with our vector tiles.

VECTOR TILE SIZE The size of a single vector tile should not be greater than 500 KB.

7

DESIGN

7.1 ARCHITECTURE

The project is divided into the `import` phase where the ETL process happens and an `export` phase where the PostGIS data is transformed into vector tiles using the `open-streets` data style. The data style depends on the database schema defined by the `import`.

Along the way many additional development `tooling` was needed for the reverse engineering process and for improving developer productivity. Everything that is not needed during the workflow but only for development or verification is in this package.

The infrastructure package contains specially configured programs as Docker images. It contains a PostGIS database, a connection pooler (pgbouncer) and a tessera based raster tile server.

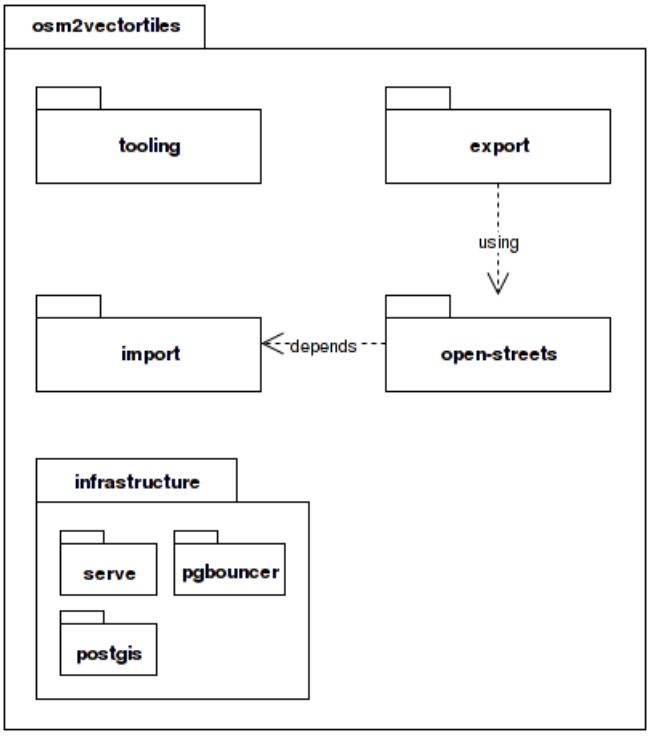

Figure 7: High level package diagram

7.1.1 *Import*

The import consists of several imports of different data sources.

IMPORT-OSM Import of the OSM planet file with a custom mapping configuration.

IMPORT-SQL Custom SQL functions to keep SQL queries in the `open-streets` data style DRY and generated SQL code for classifications.

UPDATE-SCALERANKS Custom scalerank updates from Natural Earth data for better scaleranks in places.

IMPORT-NATURAL-EARTH Import of Natural Earth data.

IMPORT-WATER Water polygons from OpenStreetMapData.

IMPORT-LABELS Custom curated labels for marine, countries and states.

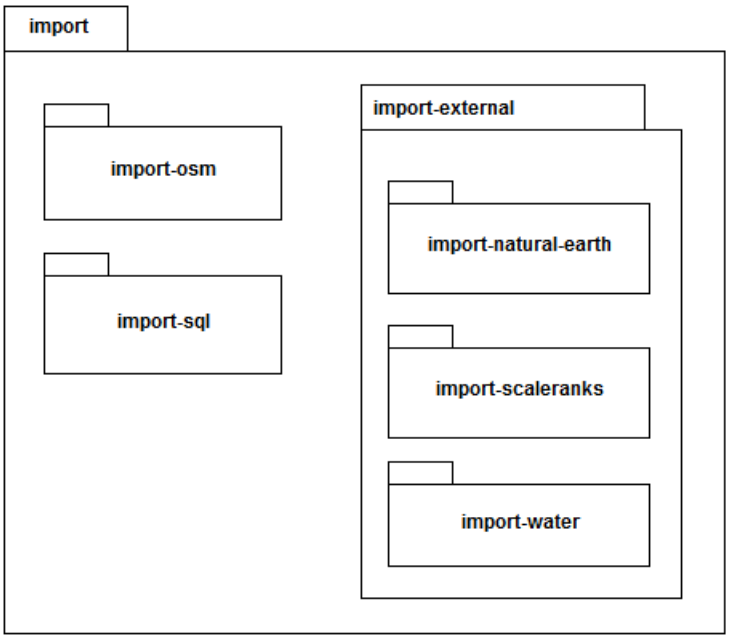

Figure 8: Import package

7.1.2 *Export*

Export can either be done locally which will process one bounding box or in parallel on several remote machines. This thesis was focused on the local export only.

OPEN-STREETS The data style project is the most essential component which pulls together all the data of different datasources to create vector tiles.

EXPORT-LOCAL A process which is using the open-streets.tm2source project to create vector tiles for a given bounding box.

7.1.3 *Tooling*

This chapter describes the various tools that were created to support the development process.

VERIFY Verify size and correctness of MBTiles files.

COMPARE Compare different vector tiles for their differences.

VISUAL-COMPARE Compare different raster tiles in an interactive map.

GENERATE-DIAGRAMS Generate mapping and layer diagrams from source code.

TEST-PERFORMANCE Load testing with Gatling for tileserver implementation.

MAPBOX-STUDIO Mapbox studio in a Docker container for working on a server.

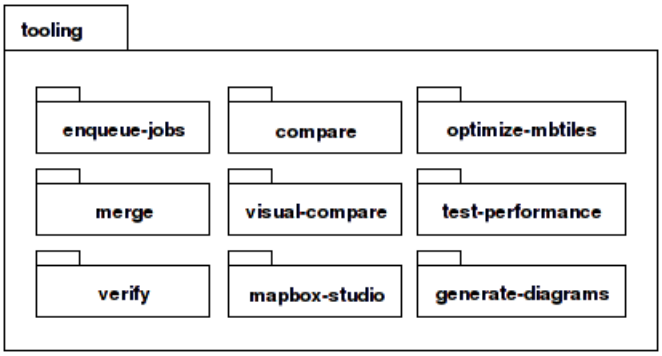

Figure 9: Tooling package

7.2 DEPLOYMENT

All workflow processes are deployed as Docker containers. The software package is usually one single process.

7.2.1 *Import*

The different import containers are attached to the `postgis` container and are part of the ETL process to import data into the database.

- `import-osm` imports data from OSM planet file.
- The `import-external` container contains the import scripts for external data sources.
- `import-sql` only imports custom SQL functions into the database and no data.
- `update-scaleranks` depends on already imported data from Natural Earth and OSM to be able to update OSM places with scaleranks from Natural Earth.

The data container pattern[5] is used in `cache` and `pgdata`. The data only containers are mounted from `import-osm` for storing cache data while importing and in `postgis` for storing the database files.

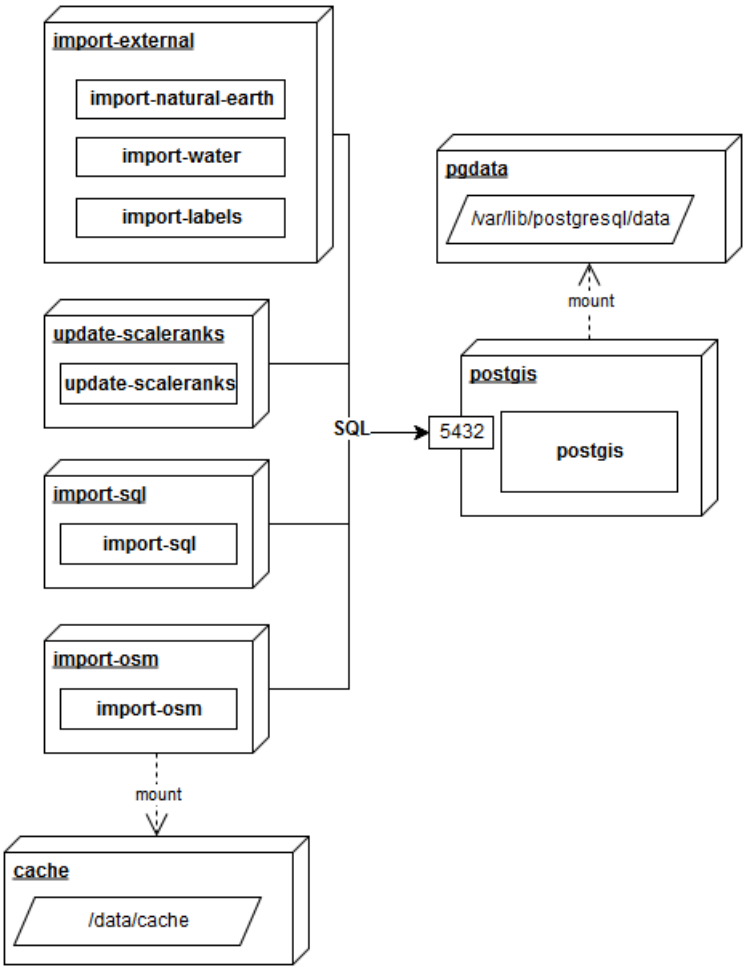

Figure 10: Import deployment diagram

7.2.2 *Export and Development Tools*

- The `export` container is responsible for rendering the vector tiles from the database and stores it in the export volume. For performance reason a connection pooler is in front of the actual `postgis` container.
- The raster tile server can serve directly from the export directory where the generated MBTiles are put.
- The `mapbox-studio` container allows editing the `open-streets` project and connects directly to the `postgis` container.
- `compare-visual` is tunneling the requests for the tiles coming from port 4001 back to the `serve` container on port 8080.

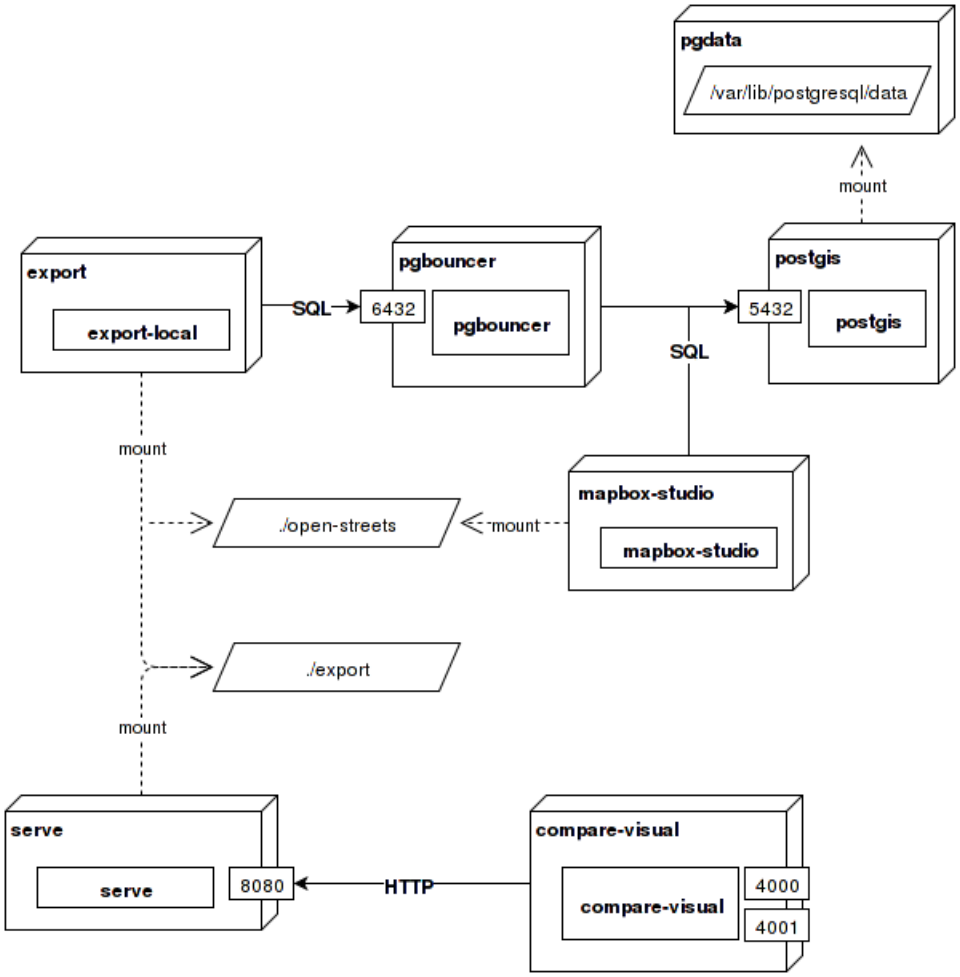

Figure 11: Export and development tools deployment diagram

7.3 WORKFLOW

7.3.1 *Import*

The import workflow is a view under the hood of the different components.

- Using the GDAL tool ogr2org[12] the Natural Earth SQLite file and custom GeoJSON files are imported into PostGIS and transformed into the right projection.
- imposm3[18] imports the planet file according to the mapping configuration into PostGIS.
- A custom python script transforms the classifications in the file into SQL functions and appends them to the existing function.sql file which is loaded into PostGIS as well.
- The update scaleranks script operates on the database only using both tables from Natural Earth and OSM to update the scalerank of places.

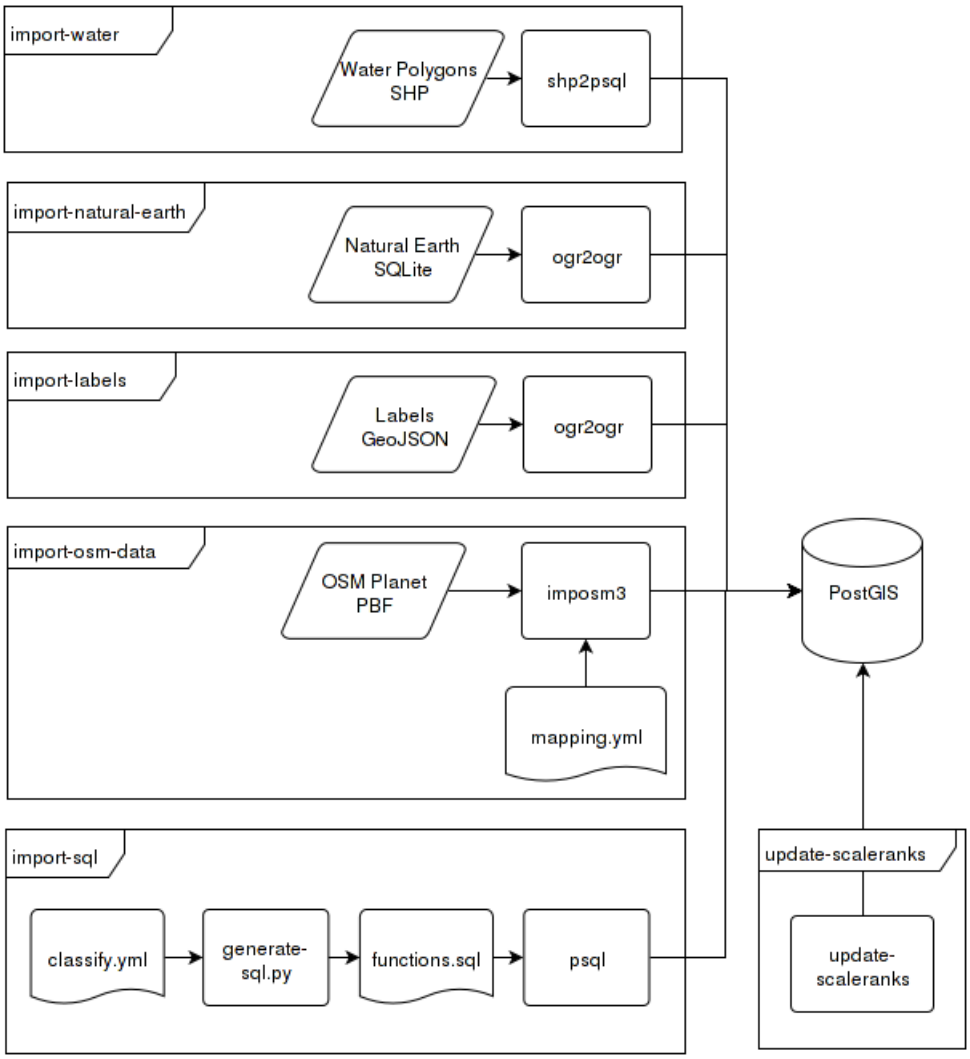

Figure 12: Import workflow from various data sources into PostGIS

7.3.2 *Export*

For generating the vector tiles the tilelive tool `tl`[19] is used which wraps around Mapnik. The data style defines all feature classes (layers) and is transformed into a Mapnik XML stylesheet by the `tilelive-tm2source` provider.
`tilelive-bridge` calls Mapnik with the generated stylesheets and hands the generated data over to `node-mbtiles` which stores the vector tiles in a MBTiles container.

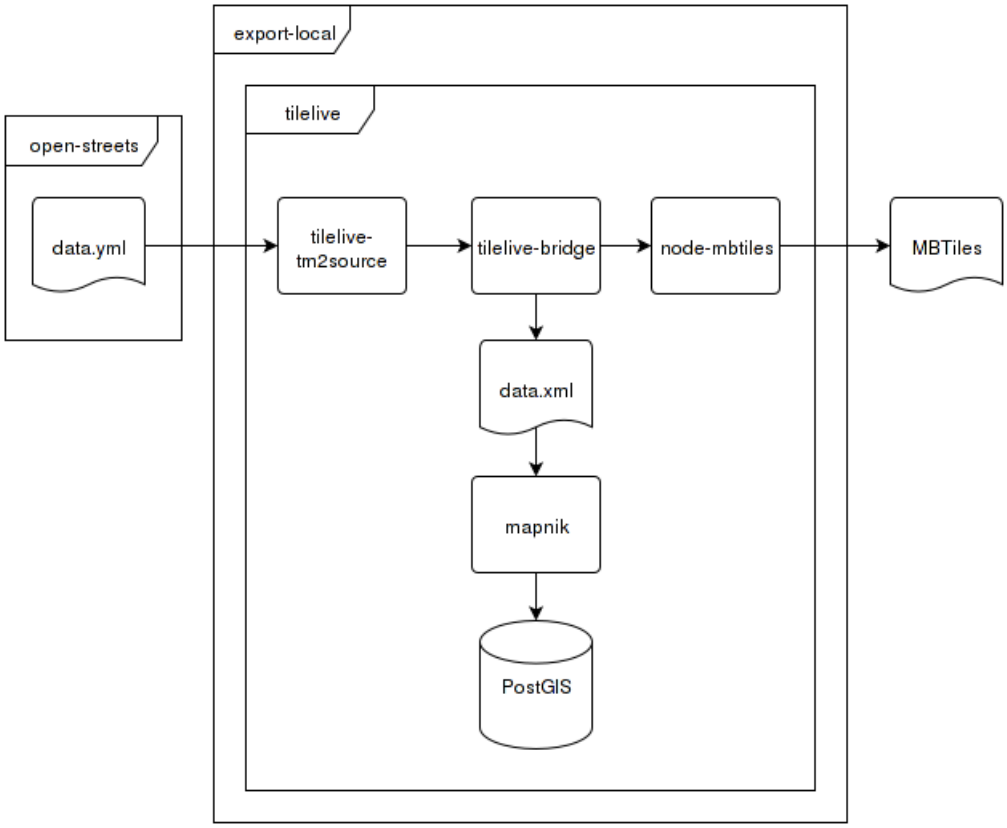

Figure 13: Export data from PostGIS to MBTiles

7.4 DATABASE SCHEMA

The schema is flat and has no relations. Each table contains information about its entity and geometry.

7.4.1 *OpenStreetMap Planet*

The cornerstone of the entire map is *OpenStreetMap* data from published snapshots from OSM Planet[71].

- Smaller extracts for single countries and continents are available from Geofabrik[8]
- For single cities or regions the Metro extracts from Mapzen[44] can be used

The data is available in the PBF[77] and OSM XML format [65]. Data in PBF format is 30% smaller and 5-6 times faster to read and write than the bzipped OSM XML version.

Imposm3 was used to import the OSM data into the PostGIS database. Selected tags[66] and their geometries are defined in an import mapping explicitly. Because imposm3 cannot match two types of polygons (e.g. polygon and point) a table was created for each geometry type.

Table Name	Geometry Type	Description
admin	linestring	Administrative boundaries
buildings	polygon	Building shapes
landusages	polygon	Human use of land
places	point	Populated settlements
roads	linestring	Roads, tracks and paths
aero_lines	linestring	Airports and aviation-related items
aero_polygons	polygon	see aero_lines
barrier_lines	linestring	Movement blocking structures
barrier_polygons	polygon	see barrier_lines
housenumbers_points	point	Address information about houses
housenumbers_polygons	polygon	see housenumbers_points
poi_points	point	Point of interest
poi_polygons	polygon	see poi_points
water_lines	linestring	Lakes and rivers
water_polygons	polygon	see water_lines

Table 1: Tables from OpenStreetMap planet file

7.4.2 *Custom Curated Labels*

The placement and importance of labels of countries, states and seas matters[1] and is important to get right. Data from the overpass API [67] is converted into GeoJSON and manually edited and enhanced with a label rank. For sea labels custom lines have been drawn to place the label along this line.

Table Name	Geometry Type	Description
custom_seas	point	Marine names
custom_countries	point	Country names
custom_states	point	State names

Table 2: Tables with custom label data

7.4.3 *OpenStreetMapData*

Certain *OpenStreetMap* data like borders and land polygons is very sensitive for change. The OpenStreetMapData[54] project takes care of a lot of issues that happen with coastlines and provide it in a convenient format. The data is checked by the OSM community and released separately.
Water polygons[55] from OpenStreetMapData were used for the ocean parts of the world. This data set ensures that the water polygons work well together with other *OpenStreetMap* data and splits big water polygons into multiple pieces for performance.

Table Name	Geometry Type	Description
osm_ocean_polygons	polygon	Ocean, seas, large lakes

Table 3: Table imported from OpenStreetMapData

7.4.4 *Natural Earth*

The Natural Earth [47] data set provides manually curated data of cultural and physical features of the world. Natural Earth data is especially useful at higher zoom levels.
The imported Natural Earth data results in more than 100 tables, but only a few are relevant for our use case. Country, state borders and large lakes are taken from Natural Earth data for the lower zoom levels.
The following data sets are used from Natural Earth:

- Label ranks of big cities[48]
- Major lakes[49]
- Country[50] and administrative[51] borders including disputed borders[52]

Table Name	Geometry Type
ne_110m_admin_0_boundary_lines_land	linestring
ne_50m_admin_0_boundary_lines_land	linestring
ne_10m_admin_0_boundary_lines_land	linestring
ne_50m_admin_1_states_provinces_lines	linestring
ne_10m_admin_1_states_provinces_lines_shp	linestring
ne_10m_admin_0_boundary_lines_disputed_areas	linestring
ne_110m_lakes	polygon
ne_50m_lakes	polygon
ne_10m_lakes	polygon

Table 4: Tables imported from Natural Earth

7.5 LAYER SCHEMA

The layer schema tries to stay compliant to the Mapbox Streets v6 layer reference [36] and Mapbox Streets v5 layer reference[37]. The detailed documentation on which database tables are mapped to which feature classes, can be found in the following sections.

Layer	Description
#landuse	Both land-use and land-cover.
#waterway	Rivers
#water	Oceans and seas
#aeroway	Aero related lines and polygons
#barrier_line	Barrier lines and polygons
#building	Building polygons
#landuse_overlay	Transparent overlays for water
#tunnel	Tunnels
#road	Roads
#bridge	Bridges
#admin	Administrative borders
#country_label	Labels of countries
#marine_label	Labels of oceans and seas
#place_label	Labels of places
#water_label	Labels of lakes
#poi_label	Labels of point of interest
#road_label	Labels of roads
#waterway_label	Labels of rivers
#housenum_label	Labels of housenumbers

Table 5: Layer descriptions of the data style

7.5.1 *Aeroways, Barriers and Landusages*

For some layers linestring and polygon data needs to be mapped into tables. The different geometries are then both rendered as vector linestrings. The landing strips of airports for example might be a linestring or polygon.

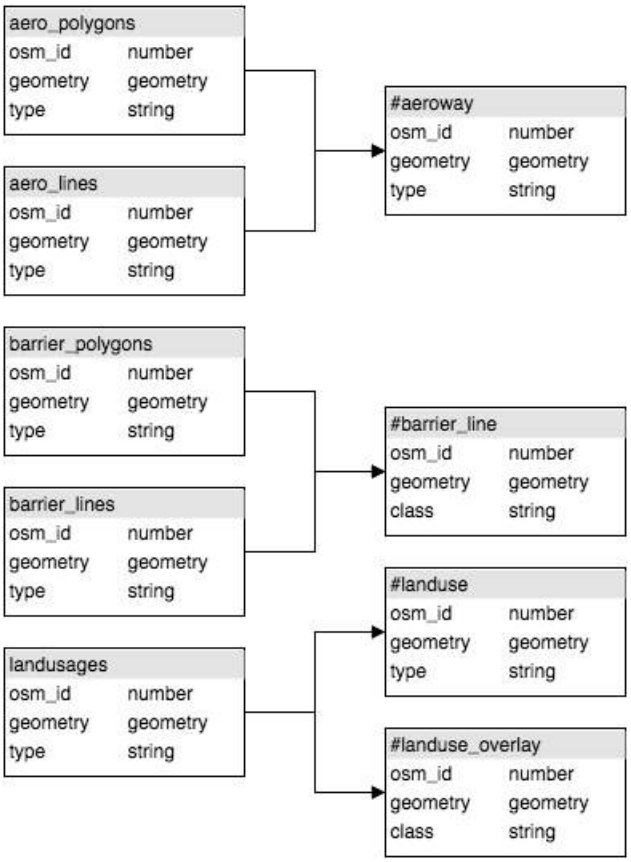

Figure 14: Layers for aeroways, barriers and landusages

7.5.2 *Administrative Borders*

The administrative area on lower zoom levels is entirely from Natural Earth data. Only at higher zoom levels where details are more important the OSM borders are rendered.
Natural Earth provides data in several generalization levels. The table with highest generalization is used on the lowest zoom levels and on higher zoom levels the less generalized tables.

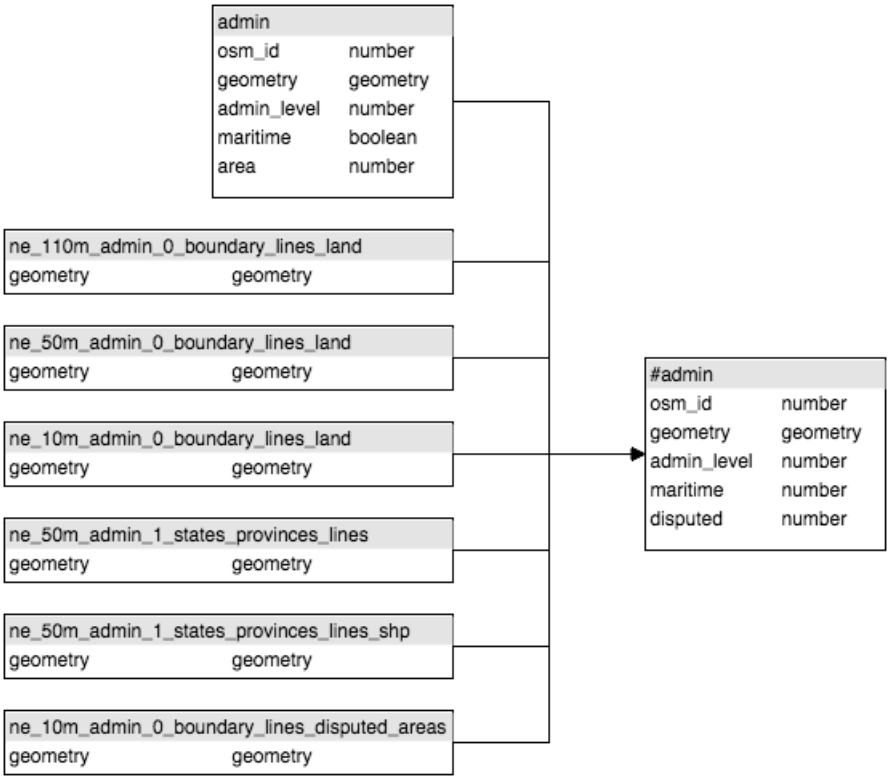

Figure 15: Layers for administrative areas

7.5.3 *Roads, Bridges and Tunnels*

Roads are split up into normal roads, tunnels and bridges after a certain zoom level. `z_order` and `layer` attributes are used to order the geometries on the right z axis. Road labels however will always contain data for tunnels, bridges and normal roads therefore one table that is filtered into different views at higher zoom levels is the best approach.

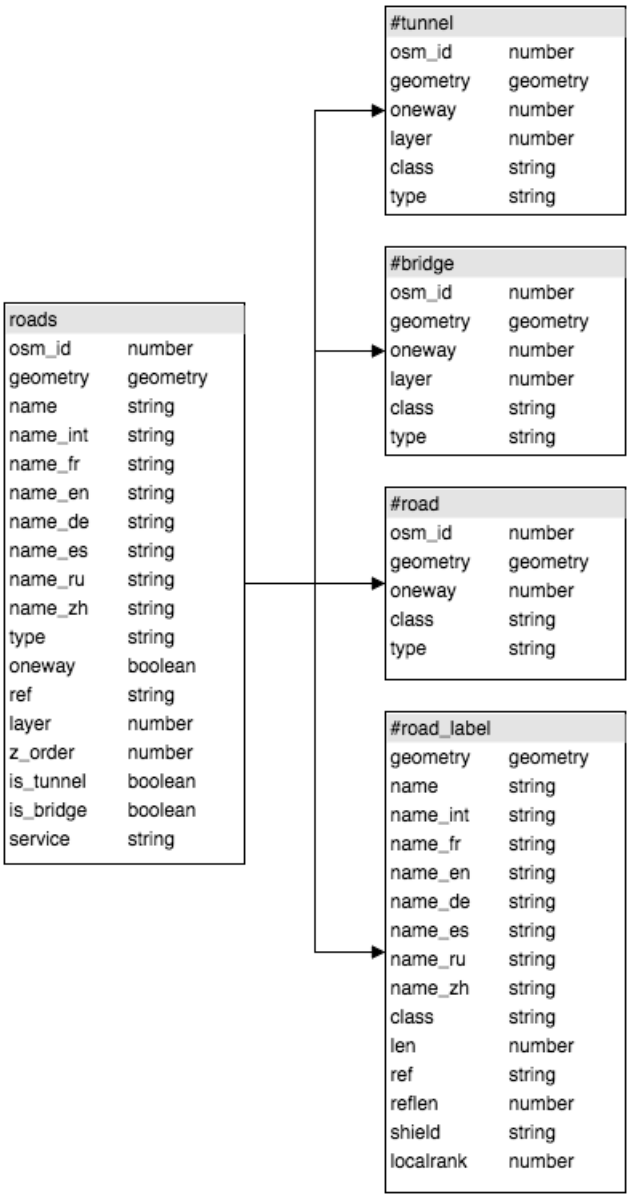

Figure 16: Layers for roads, tunnels and bridges

7.5.4 *Points of Interest*

Most POIs are in fact points, but buildings tagged with POI attributes are often polygons, which is why tables for both points and polygons are created.
The `localrank` and `scalerank` of the `#poi_label` layer are calculated from the `type` and `area` attributes. The `address` field is pulled together from the various address attributes on the tables (`street`, `housenumber`, `place`, `city`, `postcode` and `country`).

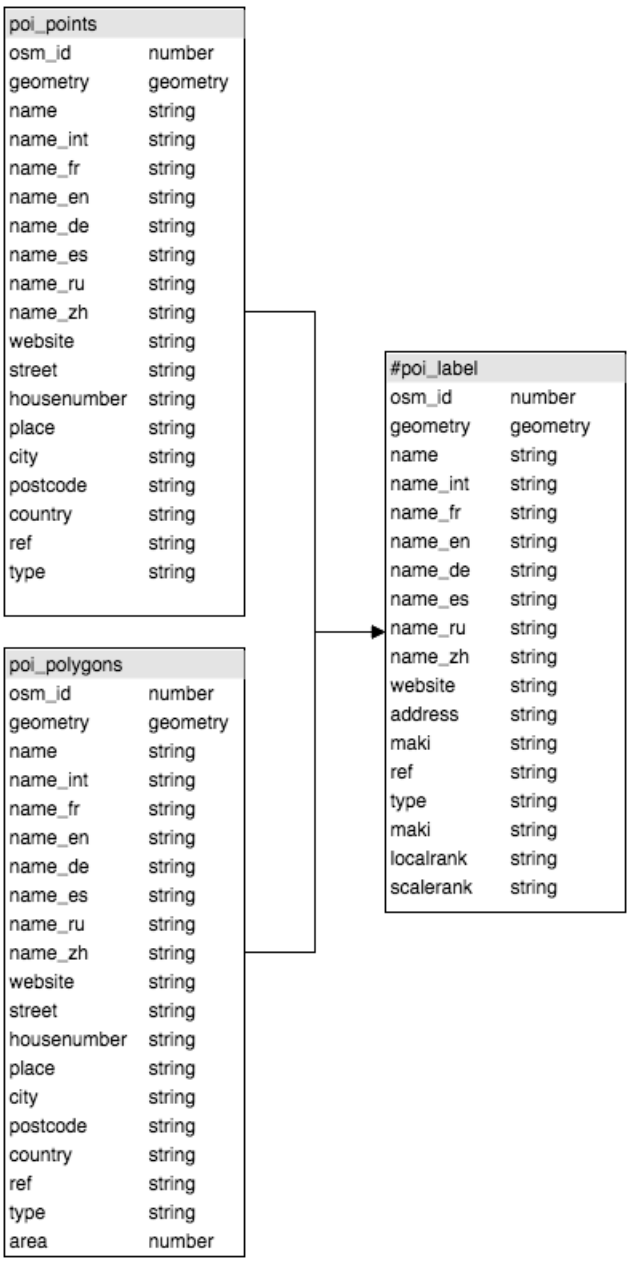

Figure 17: Point of interest label layer

7.5.5 *Water*

Water bodies for lower zoom levels are taken from Natural Earth data while lakes and rivers are from *OpenStreetMap*. Big rivers often consist out of a water polygons while smaller rivers are only water ways.

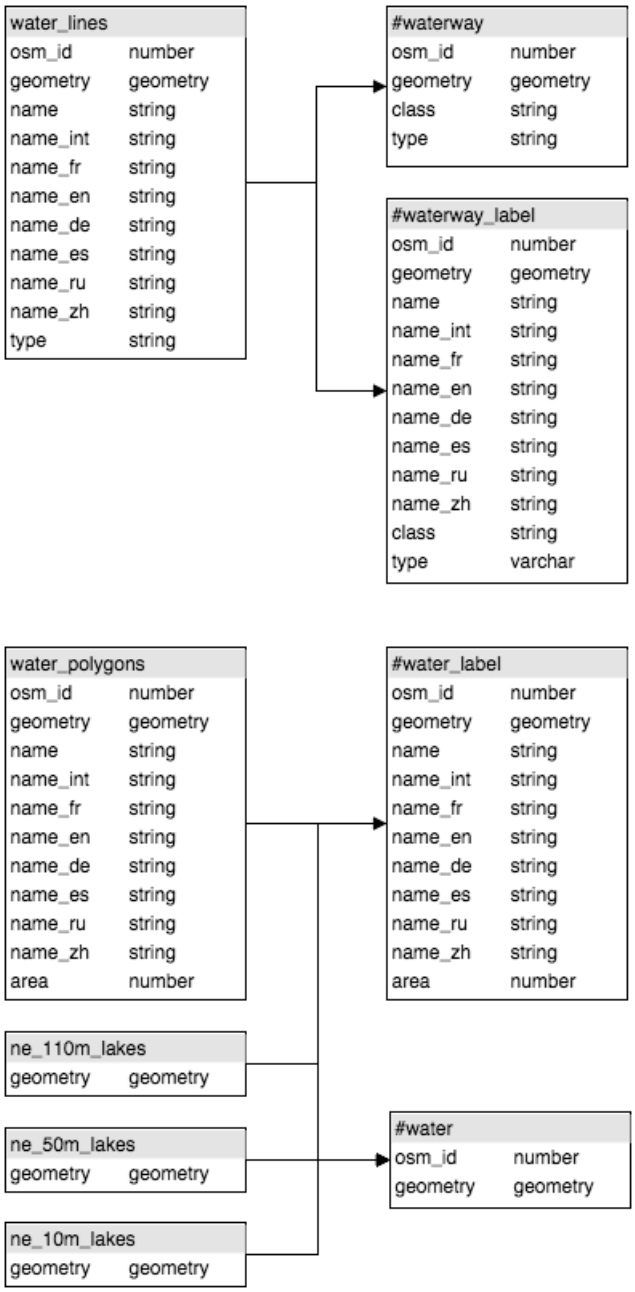

Figure 18: Water bodies and river layers

7.5.6 *Places*

Places are names of cities and villages. To calculate the importance of a city the scalerank of the most important cities from Natural Earth data is merged into the *OpenStreetMap* data set. For places that do not have a scalerank value a dynamic value is calculated based on the population.

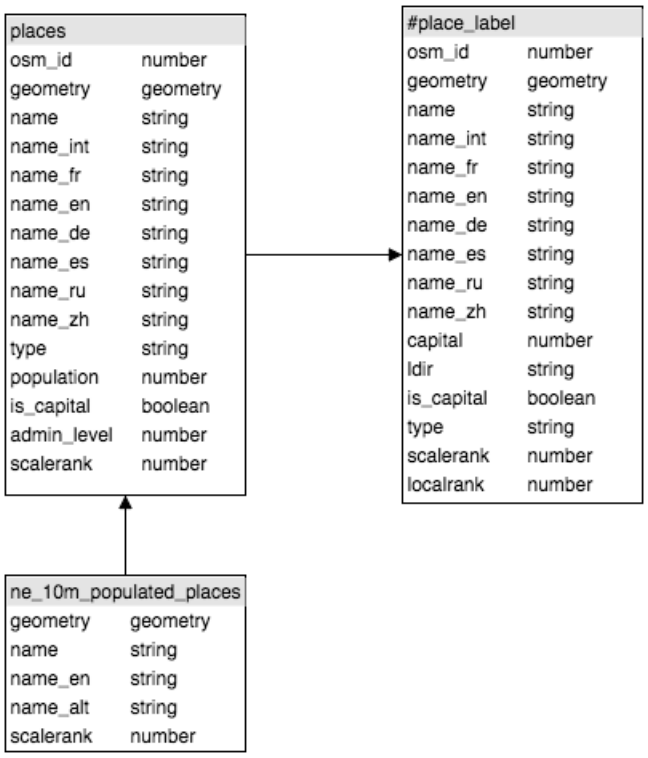

Figure 19: Place label layer

8 TECHNOLOGY EVALUATION

8.1 SPATIAL DATABASE

The OSM wiki[69] recommends PostgreSQL with the PostGIS extension for processing the data. There also exists a method[17] that circumvents using a database and directly transform OSM data into vector tiles but this does not scale for global vector tile coverage and does not support mixing additional data into the vector tiles.

Decision

PostGIS was the only viable choice due to superb tooling support for OSM data and advanced spatial query capabilities.

8.2 OSM IMPORT TOOL

As import tool the OSM community recommends imposm[27] or osm2pgsql[70]. In this section the two tools are compared with each other.

8.2.1 *Criterias*

SPEED In order to iterate fast and be able to change the data style frequently it is important that the import tool is reasonably fast and is able to import the OSM planet file in one single day.

CUSTOMIZED SCHEMA A custom import schema can be used to define the database schema and to map the OSM key/value pairs to a database table. The ability to customize the schema makes querying simpler and more performant.

DIFF UPDATES It must be possible to apply the Planet diffs [71] to continuously update the database with newer data.

EXISTING DATA STYLE PROJECTS In order to get started it is helpful to have a lot of query examples from other data style projects available.

8.2.2 *Evaluation Matrix*

Criteria	Weight	imposm	osm2pgsql
Speed	0,3	8	5
Customized Schema	0,4	7	4
Diff Updates	0,2	6	8
Existing Material	0,1	6	10
Weighted Score	1	7	5,7

Table 6: Evaluation matrix of imposm vs osm2pgsql

8.2.3 *osm2pgsql*

osm2pgsql[70] is the most commonly used import tool for processing raw OpenStreetMap data into PostGIS. The import schema is also called osm2pgsql and defines a very simple schema(line, point, polygon and roads)[72]. This results in very large tables, so it is recommended to create good indices. Osm2pgsql supports updating of the database, if the values have been stored as hstore. The schema can be adapted via the import style [73] but most projects use the default style[20] provided by osm2pgsql.

8.2.3.1 *imposm3*

Imposm is an import tool for OSM data, it is not a schema. But it defines a default schema[29], which could possibly be changed by provinding a custom mapping file. An advantage of the default schema is that it groups data thematically into tables. Which results in smaller tables and simpler queries. Imposm 3 supports updating the database from OSM diff files[26]

Decision

For the use case of this thesis it is important, that the import is efficent and that the import tool supports updating based on OSM diff files. Imposm3 is faster than osm2pgsql and supports updatability and therefore it was decided to use imposm3 for importing.

8.3 VECTOR TILE FORMAT

Vector tiles are a broad term. In this thesis vector tiles correspond to Mapbox vector tiles which is a custom open specification how vector tiles should be structured.

MAPBOX VECTOR TILES When Mapbox introduced it's geography tool Mapbox Studio in 2013 they created the *Mapbox Vector Tiles Specification* [18] which is implemented by a variety of tools and clients [14] including *Mapbox GL JS*, *Open Layers 3*, *Leaflet*, *Mapzen Tangram* and Esri [61] in the future.

GEOPACKAGE The *GeoPackage Encoding Standard* is the OGC counterpart to the *Mapbox Vector Tiles Specification* which was introduced later and is supported by QGIS, ESRI and GDAL.

GOOGLE MAPS Google Maps is using vector tiles since 2010 under the hood and was the first provider implementing this. Styling is limited and the format proprietary.

Decision

Because one of the main requirements of this project was to provide Mapbox Streets compatible vector tiles and Mapbox provides very good tools to handle vector tiles, there was no other choice than going with Mapbox's implementation of vector tiles.

8.4 VECTOR TILE SERVER

Next to the vector tiles for Switzerland, the second deliverable is a basic vector tile server. The goal is that a non technical person can get started quickly with the custom vector tiles.

Unlike the choice for a spatial database or OSM import tool there is no typical setup method of a raster tile server using vector tiles under the hood. Most people in the Open Source community build their own specific tile server setup.

8.4.1 *Tessera*

Tessera[15] is a Node.js[10] webserver, which is using Mapbox tilelive[16] modules to read vector tiles and generates raster tiles.

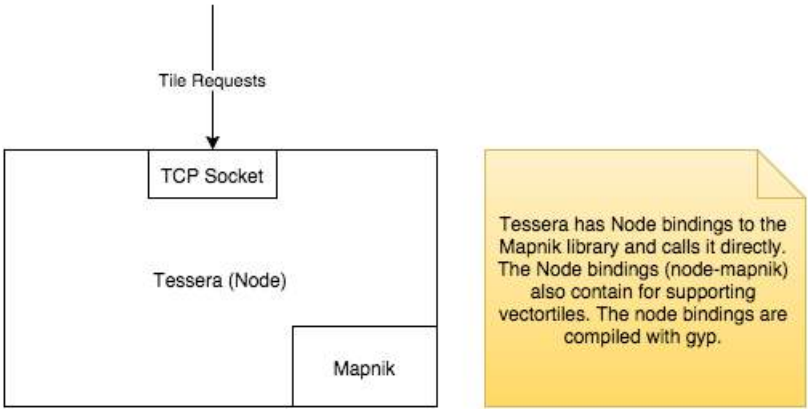

Figure 20: Architecture Diagramm Tessera

Load tests were performed to measure server performance with tessera.
Infrastructure: AWS EC2 t2.micro instance(1 GB Memory / 1 Core)
Task: Zooming in from zoom level 10 to 22
Conditions: 50 concurrent users

The results of the load test is shown below.

```
---- Global Information --------------------------------------------------------
> request count                                      10350 (OK=10348  KO=2     )
> min response time                                     50 (OK=50     KO=60010 )
> max response time                                  60335 (OK=56103  KO=60335 )
> mean response time                                  2138 (OK=2127   KO=60172 )
> std deviation                                       3023 (OK=2914   KO=162   )
> response time 50th percentile                       1115 (OK=1114   KO=60172 )
> response time 75th percentile                       3127 (OK=3125   KO=60253 )
> mean requests/sec                                75.919 (OK=75.904 KO=0.015 )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                          4545 ( 44%)
> 800 ms < t < 1200 ms                                 756 (  7%)
> t > 1200 ms                                         5047 ( 49%)
> failed                                                 2 (  0%)
---- Errors --------------------------------------------------------------------
> java.util.concurrent.TimeoutException: Request timed out to ec      2 (100.0%)
2-52-30-184-45.eu-west-1.compute.amazonaws.com/52.30.184.45:80...
```

With 50 concurrent users tessera was still able to respond to all requests in less than 1200 ms.

8.4.2 *OpenStreetMap "Standard" Tile Server*

A proven set up for generating raster tiles directly from PostgreSQL with Mapnik consists of an Apache webserver and a custom Apache module mod_tile[68]. This approach was around before vector tiles were proposed.

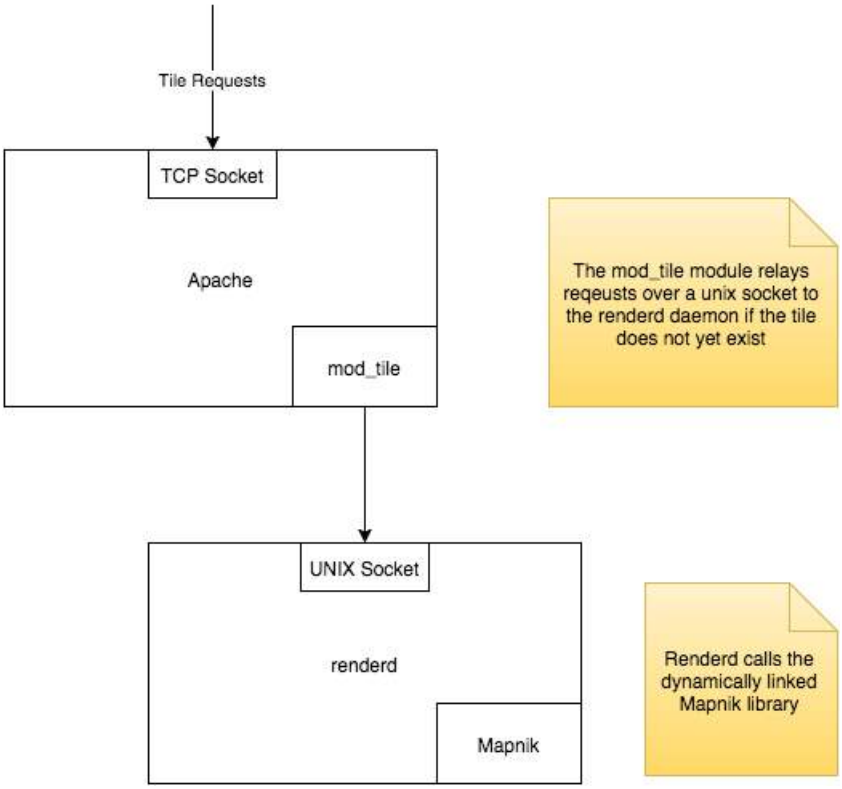

Figure 21: Architecture Diagramm OSM Standard tile server

It is theoretically possible to implement a raster tile server on top of Mapnik and renderd using C++. Instead of using a spatial database as source for the rendered raster tiles a datasource binding would need to be implemented to read the vector tiles and hand them over to the Mapnik renderer.

Decision

The plan was to perform the load test on each version of the vector tile server and then make the decision based on the results. Due to the fact, that the test results of tessera where good enough for the thesis use case and it didn't take much time to implement, it was decided to use tessera as raster tile server. If somebody needs a high-performance tile server, one should probably think about the second variant.

9 IMPLEMENTATION

The implementation chapter explains the aspects of the different solutions in greater detail.

9.1 MAPPING

Imposm3 requires a custom mapping file to decide which tags and geometries are mapped into which table.
The example data mapping[28] below defines that geometries of the type polygon with the key value pairs `natural=wood`, `natural=land` and `tourism=zoo` should be mappend into the `landusages` table.

```
tables:
   landusages:
     type: polygon
     mapping:
       natural: [wood, land]
       tourism: [zoo]
```

Instead of mapping all values of a certain key an explicit mapping strategy has been chosen. Each tag was explicitly defined in the mapping. This brings the benefit of knowing exactly what tags are in the database which makes querying and filtering easier.

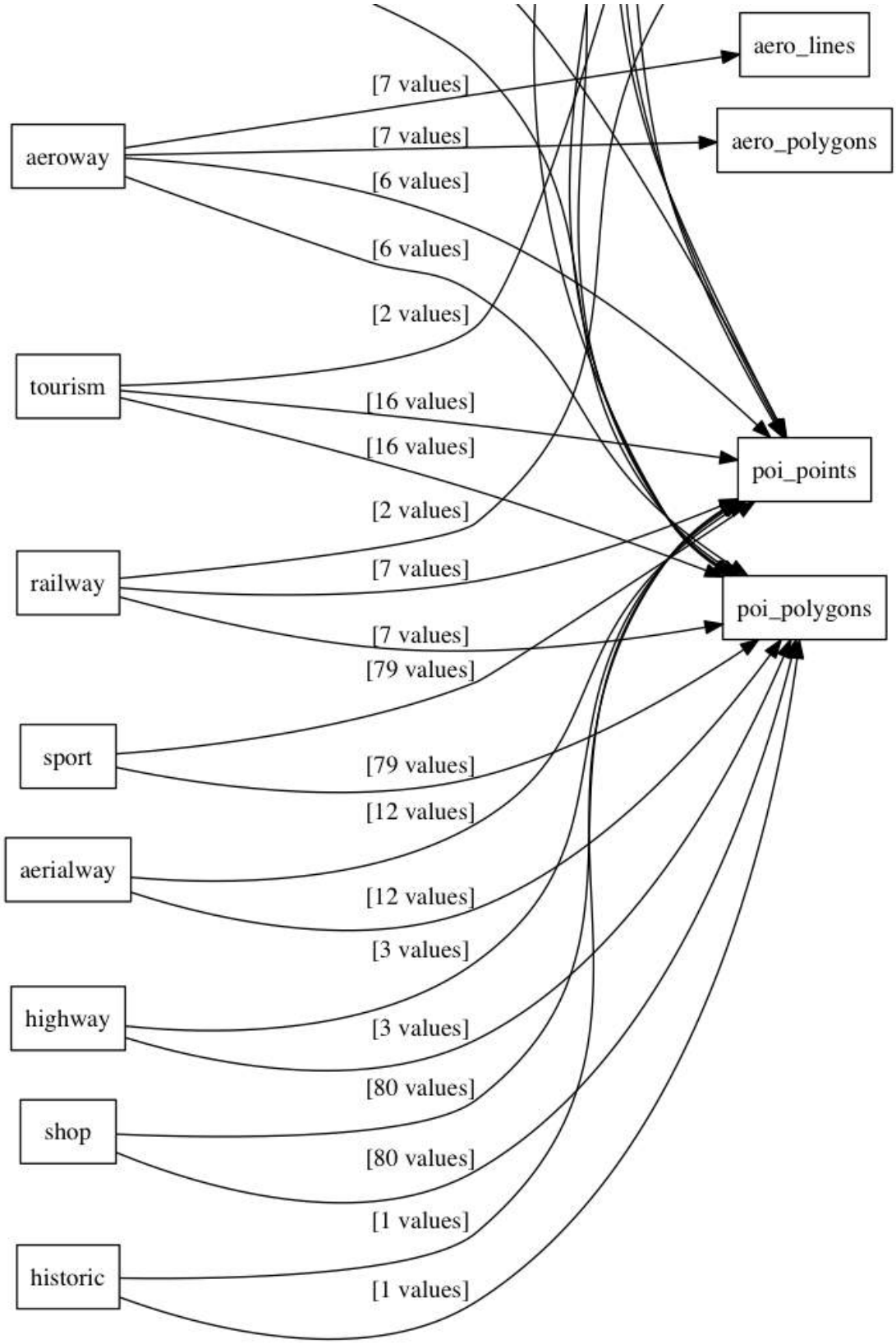

Figure 22: Mapping of tags to tables

9.2 DATABASE SCHEMA

This section describes the implementation details of the database schema.

9.2.1 *OSM id*

All tables and views that are derived from *OpenStreetMap* data have the original id[74] to trace back the data and be able to query a distinct *OpenStreetMap* feature in a visual style.

9.2.2 *Translations*

All label features contain several translations. The translations are directly mapped from the suffixed language code [75]. If a field is not available in the language, the local name is used.

Field	Description
name	Local name
name_en	English
name_es	Spanish
name_fr	French
name_de	German
name_ru	Russian
name_zh	Chinese

Table 7: Translations of name field

9.2.3 *Type and Class*

The `class` field is equivalent to the feature class while the `type` field is the OSM value of the feature.

9.3 CLASSIFICATION

The *OpenStreetMap* tagging schema has developed into a complex taxonomy of real-world feature classes and objects. [25, p. 15]. Map designers don't want to design for each tag specifically which is why Mapbox and other providers abstract distinct tags into feature classes. For example a map designer that wants to style agricultural areas does not care what type of field it is. Mapping tags into categories cannot be automated and there is no standard defined: therefore the mapping is handmade.

Key	Value	Class	Type
landuse	farm	agriculture	orchard
building	farm	agriculture	farm
landuse	farmland	agriculture	farmland
landuse	farmyard	agriculture	farmyard
landuse	allotments	agriculture	allotments
landuse	vineyard	agriculture	vineyard
landuse	vineyard	agriculture	plant_nursery

Table 8: Classification of landuse feature class

9.3.1 *Classification Format*

The classicifcations are written in a YAML based format where each key in `classifications` denotes the classification name. The keys within each classification (e.g. `driveway`, `main`) contain the class name. The values in each class name (e.g. `primary`, `primary_link` represent the OSM values that need to be matched. Only the value of a OSM tag is taken into account for deciding the feature class. The key of the tag is not matched.

```
classifications:
  road:
    highway:
    - motorway
    - motorway_link
    - driveway
    main:
    - primary
    - primary_link
    - trunk
    - trunk_link
    - secondary
    - secondary_link
    - tertiary
    - tertiary_link
```

9.3.2 *Code Generation*

The `generate_sql.py` script reads the classification format and generates immutable SQL functions from the YAML source. These functions can then be used in the layer queries.
The example in the above listing will result in the following function.

```
CREATE OR REPLACE FUNCTION classify_road(type VARCHAR)
RETURNS VARCHAR AS $$
  BEGIN
    RETURN CASE
      WHEN type IN ('motorway','motorway_link','driveway') THEN 'highway'
      WHEN type IN ('primary','primary_link',
                    'trunk','trunk_link',
                    'secondary','secondary_link',
                    'tertiary','tertiary_link') THEN 'main'
    END;
  END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

Classifications are then baked into vector tile attributes of geometries.

```
SELECT
  geometry,
  classify_road(type) AS class,
  type AS type
FROM osm_roads
```

9.4 RELATIVE IMPORTANCE

To reduce label density on lower zoom levels, but still contain all data in e.g. zoom level 14, the `localrank` attribute indiciates how important a label is compared to the labels in its neighbourhood.

9.4.1 *Calculating Rank*

9.4.1.1 *Order Features by their Types*

In the best case scenario a function would rank each point of interest class. Due to the limited time only the most important features were explicitly ranked.

```
CREATE OR REPLACE FUNCTION localrank_poi(type VARCHAR) RETURNS INTEGER
AS $$
BEGIN
  RETURN CASE
    WHEN type IN ('station', 'subway_entrance', 'park',
                  'cemetery', 'bank', 'supermarket', 'car',
                  'library', 'university', 'college', 'police',
                  'townhall', 'courthouse') THEN 2
    WHEN type IN ('nature_reserve', 'garden', 'public_building') THEN 3
    WHEN type IN ('stadium') THEN 90
    WHEN type IN ('hospital') THEN 100
    WHEN type IN ('zoo') THEN 200
    WHEN type IN ('university', 'school', 'college', 'kindergarten') THEN 300
    WHEN type IN ('supermarket', 'department_store') THEN 400
    WHEN type IN ('nature_reserve', 'swimming_area') THEN 500
    WHEN type IN ('attraction') THEN 600
    ELSE 1000
  END;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

9.4.1.2 *Calculate Rank across Grid*

The rank is calculated across a grid of 128 pixels. The most important features from the `localrank_poi` function will also be the most relevant POIs.

```
SELECT
  geometry,
  rank() OVER (PARTITION BY LabelGrid(geometry, 128 * !pixel_width!)
               ORDER BY localrank_poi(type) ASC) AS localrank,
FROM osm_poi
```

9.5 POSTGRESQL PERFORMANCE

Most of the heavywork is done on the database side to respond to all SQL queries made by Mapnik in the fastest way possible.

9.5.1 *Tuning*

The PostgreSQL default parameters do not deliver good performance for stronger machines.[82] For the different database machines the PgTune[56] calculator has been used to determine good cache and buffer sizes for data warehouse style computing.

Example configuration for a host with 50 GB of memory.

```
max_connections = 20
shared_buffers = 12800MB
effective_cache_size = 38400MB
work_mem = 320MB
maintenance_work_mem = 2GB
checkpoint_segments = 128
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 500
```

For speed up the import, the transactional features of PostgreSQL have been disabled. [2].

```
bgwriter_lru_maxpages = 0
wal_level = minimal
fsync = off
synchronous_commit = off
full_page_writes = off
wal_log_hints = off
```

9.5.2 *Indizes*

For each table a GiST[57] index on the geometry and a clustered geohashed index was created. This helped to make lookups that check if a geometry is in a certain tile faster. Especially the clustered index hash helped in speeding up the queries.

```
"osm_places_geom" gist (geometry)
"osm_places_geom_geohash btree" (
    st_geohash(st_transform(st_setsrid(box2d(geometry)::geometry, 3857), 4326))
) CLUSTER
```

9.6 DATA STYLE

The data style is a description of all the feature classes such as landuse, water or roads. This description was invented by Mapbox.

The format of a data style looks like this:

```
_prefs:
  disabled: []
  inspector: false
  mapid: ''
  rev: ''
  saveCenter: true
attribution: ''
center:
  - 21.7969
  - 34.6694
  - 3
description: Open Streets
Layer:
    # Layer definitions come here
maxzoom: 14
minzoom: 0
name: Open Streets
```

The `center` attribute defines the default position of the map when the data style is opened with Mapbox Studio Classic. In this case the default position is set to the coordinates `21.7969, 34.6694` on zoom level 3. The `layer` attribute defines all the layers that are present in a vector tile. A detailed description follows in the next section. The max- and minzoom attributes define the range in which vector data is available.

9.6.1 *Layer Definition*

A layer definition describes a view on the data. It can consist of multiple data sources. In the figure below the layer is a view and the definition of this view is a graphic definition.

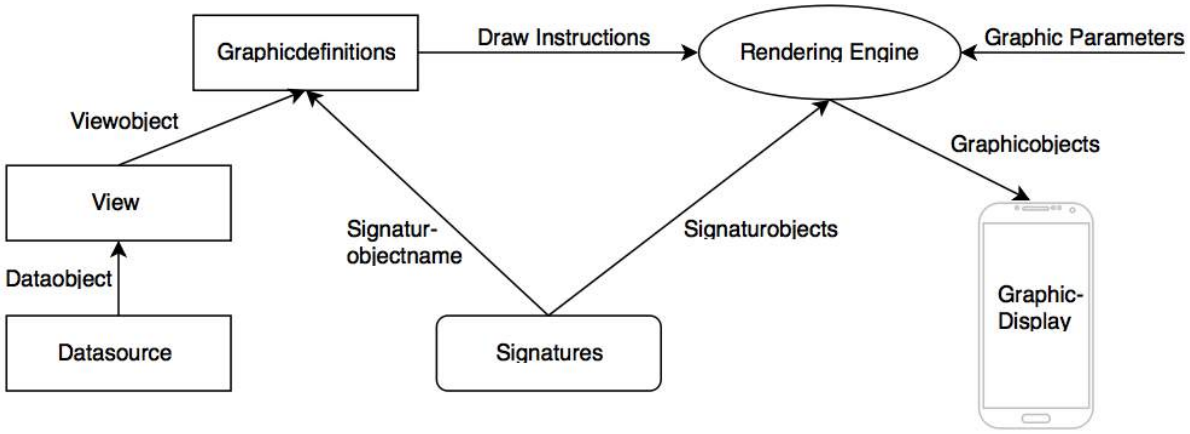

Figure 23: General graphic definition

A layer definition looks like this:

```
- id: landuse
Datasource:
    extent: -20037508.34,-20037508.34,20037508.34,20037508.34
    host: db
    port: 5432
    user: osm
    password: osm
    dbname: osm
    key_field: osm_id
    table: |-
        (
            SELECT osm_id, class, type, geometry
            FROM osm_landusages
            WHERE geometry && !bbox!
            AND z(!scale_denominator!) > 5

        ) as data
    type: postgis
fields:
    osm_id: Number
    class: String
    type: String
properties:
    "buffer-size": 4
```

The layer definition consist of the `id` (layername), datasource, fields and the properties. The data source in this case defines how the PostGIS database can be accessed and which SQL query needs to be executed. An alternative data source could also be a simple file (GeoJson, Shapefile, SQLite database, GeoTIFF, KML, GPX or CSV).

9.6.1.1 *Buffers*

The buffer value on a layer defines how many pixels around each tile will be included. It is necessary to ensure correct rendering across tile boundaries. This value is individual for each layer and depends on the type of data. Buffers for layers containing labels should have a large buffer size such as 128 pixels, whereas a layer like landuse does only need a buffers size of 4 pixels. In general, the buffer size should be set to the minimum to keep the size of the vector tiles as low as possible.[39]

Figure 24: Example for buffer values

The figure above is a good example to see the result of the buffer value. Apart from the rivers there is no other data, therefore the rivers must have a larger buffer value than the other layers.

9.6.1.2 *Overzooming*

The min- and maxzoom values define on which zoom levels data is available. This does not mean that it is not possible to zoom deeper than the maxzoom value. Overzooming defines to term of displaying data at higher zoom levels[40]. This allows to show data on higher zoom levels, without generating vector tiles for these zoom levels. Mapbox has defined a rule of thumb for vector tiles: vector tiles are useful for about 4-6 levels of overzooming. A vector tile on zoom level 10, can be stretched out up to zoom level 14 or 16.

9.6.1.3 *Layer Ordering*

The order in which the layer are defined in the data style is equal to the order they are stored in the vector tiles. The layer at the top of the layer definition will be drawn first next the second layer and so on. The layer at the bottom is drawn on top of all the other layers.

9.7 ZOOM LEVEL REFERENCE

The zoom level reference helps to see which feature class is included on which zoom level. The zoom levels are in the same order as they get drawn. The feature class `landuse` is at the bottom and `housenum_label` is on top of all the others.

	z0	z1	z2	z3	z4	z5	z6	z7	z8	z9	z10	z11	z12	z13	z14
landuse						x	x	x	x	x	x	x	x	x	x
waterway									x	x	x	x	x	x	x
water	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
aeroway													x	x	x
barrier_line															x
building														x	x
landuse_overlay								x	x	x	x	x	x	x	x
tunnel												x	x	x	x
road						x	x	x	x	x	x	x	x	x	x
bridge													x	x	x
admin	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
country_label		x	x	x	x	x	x	x	x	x	x	x	x	x	x
marine_label		x	x	x	x	x	x	x	x	x	x	x	x	x	x
state_label					x	x	x	x	x	x	x	x	x	x	x
place_label					x	x	x	x	x	x	x	x	x	x	x
water_label											x	x	x	x	x
poi_label															x
road_label									x	x	x	x	x	x	x
waterway_label									x	x	x	x	x	x	x
housenum_label															x

Table 9: Feature classes on different zoom levels

9.8 REVERSE ENGINEERING PROCESS

One of the main requirements of the project was to make the vector tiles compatible with the Mapbox Streets vector tiles[36].
This requirement has not been defined since the beginning, it evolved during the construction of the first prototype. Therefore a lot of time has been spent making the vector tiles as similar as possible to the ones of Mapbox.
The sections below describe the tools and methods that were used to achieve the goal of Mapbox Streets compatibility.

9.8.1 *Vector Tile Format*

To better understand what the vector tile compare tool does, a high level introduction to the vector tiles format will be given below.

A vector tile can consist of one or more named layers which contain one or more features[18]. A feature consists of attributes and a geometry (point, linestring or polygon). Attributes are represented as a dictionary of key-value pairs.

Below is an example of a vector tile with two layers `water` and `admin`. The water layer has the attribute key "osm_id" and value 0. If somebody would compare this example with the specification[18], one could think this is not a specification conform vector tile. The example below is a compressed vector tile. More information on what compression methods are applied can be found in the specification.

Mapbox Streets v6 vector tile for `0/0/0`.


```
{
  "layers": {
    "water": {
      "version": 1,
      "name": "water",
      "extent": 4096,
      "length": 18,
      "_pbf": {
        "buf": [26,143,32,10,5,119,97,1],
        "pos": 51410,
        "length": 51410
      },
      "_keys": ["osm_id"],
      "_values": [0],
      "_features": [11,3474,3499,3530,3561,3584]
    },
    "admin": {
      "version": 1,
      "name": "admin",
      "extent": 4096,
      "length": 1447,
      "_pbf": {
        "buf": [26,143,32,10,5,1],
        "pos":51410,
        "length":51410
       },
      "_keys": ["admin_level","disputed","iso_3166_1","maritime"],
      "_values": [2,0,"FR",1],
      "_features": [4126,4152,4377,4403,4429,4455,4481,4507,4533]
    }
  }
}
```

9.9 QUALITY ASSURANCE TOOLS

During the prototyping phase a number of quality assurance tools were built to help the contributors track the progress of Mapbox Streets compatibility.

9.9.1 *Vector Tile Compare*

The Vector Tile Compare tool analyzes vector tiles and outputs interesting information like layers and attributes. The output was generated for the same vector tiles of Mapbox Streets and the resulting vector tiles from osm2vectortiles (Open Streets). The results where then uploaded to a GitHub repository and the branch compare feature was used to compare them.

```
-name: mapbox-streets-v5                    +name: open-streets-2015-10-31
 tile: 2/2/0                                 tile: 2/2/0
-features: 206                              +features: 493
-layers: 3                                  +layers: 1
 classes: 0                                  classes: 0
 types: 0                                    types: 0

-#admin (features: 157)                     +#water (features: 493)
-       [admin_level]
-       [disputed]
-       [maritime]
-       [osm_id]
-        no class
-
-#country_label (features: 47)
-       [name]
-       [name_de]
-       [name_en]
-       [name_es]
-       [name_fr]
-       [osm_id]
-       [scalerank]
-        no class
-
-#water (features: 2)
        [osm_id]                                    [osm_id]
         no class                                    no class
```

Figure 25: Compare of Mapbox Streets v6 and Open Streets on 31.10.2015

This resulted in a indicator of which layers are shown on which zoom level and what attributes are contained in a layer.

9.9.2 *Visual Compare*

The vector tile comparison was good to ensure that exactly the same data is present on the same zoom level as Mapbox Streets. But when the contributors started to visually compare the map with the map of Mapbox Streets, big differences appeared. The decision was made to build a compare tool that works visually to cover up bugs like these.

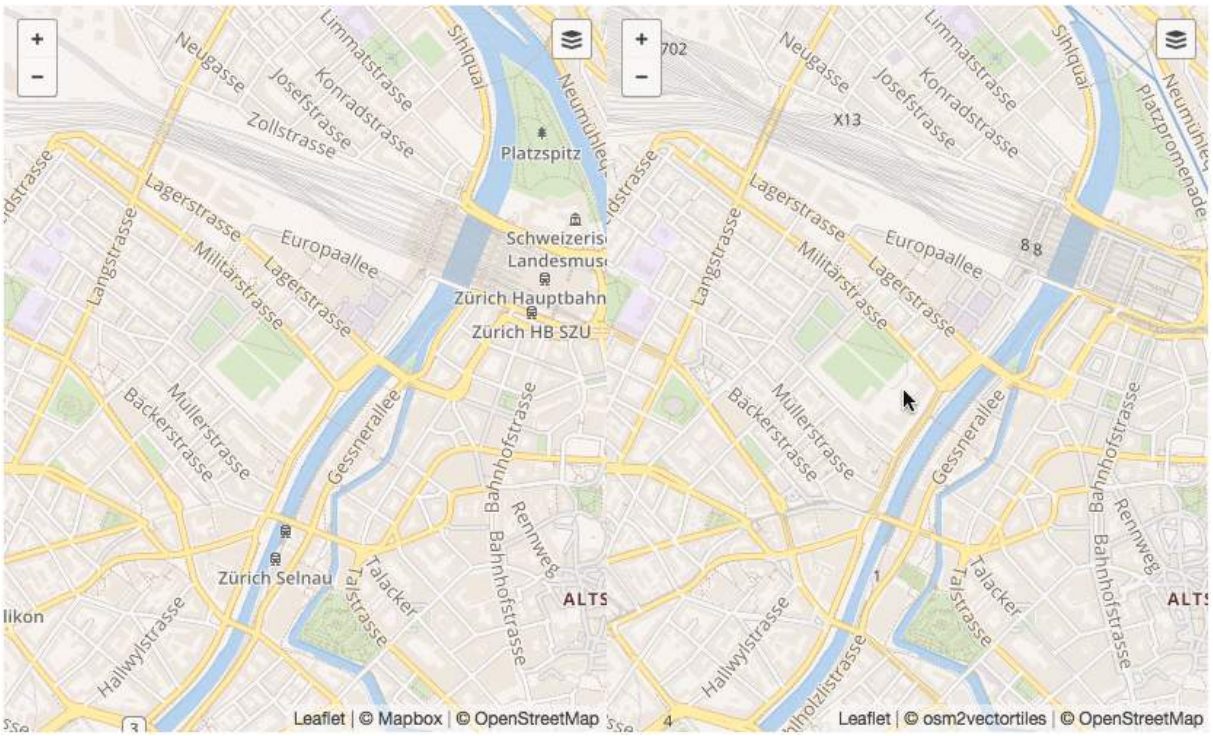

Figure 26: Visual Compare of Mapbox Streets and Open Streets

The figure above shows a screenshot of the Visual Compare tool. On the left side is Mapbox Streets and on the right side our Open Streets map. They both use the same visual style (OSM Bright 2[21]). When you zoom in on the right side of the map, it will automatically zoom in on the left side. This tool was very helpful to find all the differences.

9.10 IMPROVEMENT PROCESS

This section describes the improvement process used to eliminate visual differences.

1. When a visual difference was identified with the Vector Compare or Visual Compare tool, the contributors checked if the missing data is included in the import mapping.
2. If this was not the case, the TagFinder tool[76] was used to find the right OSM key-value pair of the missing item.
3. Next the key-value pair was included into the import mapping and the OSM data was reimported again.
4. To see the item on the map, the SQL query in the data style needs to be altered to fetch the added item.
5. Now the missing item should appear on the map.

This workflow was executed within Mapbox Studio Classic[41] where the map can be rerendered on the fly after a query has been altered.

9.11 LIMITATIONS

TILESERVER The tessera based tileserver does not support more than 50 concurrent users and is not meant for production use without a caching reverse proxy in front.

RENDERING WORKFLOW The rendering process is scalable by deploying multiple databases and vector rendering processes on very powerful hosts. For adequate rendering performance at least 16GB (or better 50GB) of memory is required to provide fast database access. Supporting an additional zoom level 15 would require a significant additional effort in computing capacity for rendering the additional tiles.

UPDATABLE VECTOR TILES The vector tiles are based of an OSM planet file at a specific point in time. It is essentially a snapshot of the OSM data. Therefore the Swiss OSM community requested updatable vector tiles based on the diff files.

BASE MAP The resulting vector tiles allow creating an alternative base map, which is customizable. The vector tiles are not meant to be queried and do not support custom overlays. Additional map features need to be implemented with the help of other libraries.

OSM DATA The vector tiles contain only a subset of all OSM data. The standard OSM basemap could not easily be replaced with our vector tiles as data source because of this limitation.

10 RESULTS AND FUTURE

10.1 RESULTS

The result of this study thesis are described in Part 1, *section 4.1*.

10.2 FUTURE DEVELOPMENT

The first version of the rendered vector tiles has shown, that it is possible to get very close to compatibility with Mapbox Streets in a reasonable amount of time. There are still some quirks, which need to be ironed out, but we created a good foundation for future development. The two sections below describe small improvements and new features, which will be implemented in the bachelor thesis.

10.2.1 *Small improvements*

LABELS The scalerank of the place, marine, state and road labels should be improved. Th current implementation is good for the moment, but still not equal to Mapbox Streets.

STATE LABELS State labels are used to display states of big countries like USA, Russia or China. More state labels could be added.

COUNTRY BOUNDARIES A mix of Natural Earth and OSM data is used for the administrative boundaries. This layer is using both data sources, because OSM has some issues with marine borders and simplified borders on lower zoom levels. Using Natural Earth data alone is not suitable, because the data is generalized. Therefore it does not look good on higher zoom levels. For the next release a better data source should be chosen for administrative boundaries.

COUNTRY NAME TRANSLATION RULE Mapbox has defined a fallback rule[36] for the country names. Due to time limits this behavior could not be implemented and should be implemented in a future release.

SPECIAL MAKI ICONS FOR US ROAD LABELS Mapbox uses different maki icons[34] for the road labels in the US.

POSSIBLITY OF MAPBOX TERRAIN VISUAL STYLE Check if all the data required for a Mapbox style like Mapbox Terrain[35] is included in the vector tiles

EXCLUDE WATER IN RENDERING PROCESS When the vector tile rendering process was implemented, it was realized that a lot of rendering time could be saved when the water (no data) is excluded. It turned out, that deciding if a vector tiles will have no data, is not an easy problem. At the current stage of the project everything is rendered and the "empty" vector tiles are removed afterwards.

10.2.2 *New Features*

VECTOR TILES OF ENTIRE WORLD One of the main targets of the bachelor thesis is to render the vector tiles of the entire world. This brings new challenges like scaling the rendering infrastructure.

UPDATE VECTOR TILES A big request of the Swiss OSM community was to provide updated vector tiles based on the diff[71] files. This also requires detecting dirty tiles in the already rendered vector tiles because one update can affect multiple tiles across multiple zoom levels.

FASTER TILE SERVER There is no robust and scalable raster tile server based on vector tiles available yet. This may even be a separate project.

GEOGRAPHIC NAME SEARCH To fulfill the long term goal of this project to provide an offline map a basic geographic name search could be implemented.

11 PROJECT MANAGEMENT

11.1 SOFTWARE DEVELOPMENT PROCESS

An agile approach based on RUP has been used as the process model of this project. At the kickoff meeting the schedule with the milestones was defined and the first tasks were assigned to the alpha milestone. In frequent meetings the progress of these tasks were tracked and new tasks were created. The work was divided into two to three week iterations with a deliverable version at the end of each iteration.
Constant evolution has been favored over long term planning and this approach has proven itself for this type of work, where unexpected problems are the rule not the exception.

11.1.1 *GitHub*

GitHub was used for planning and tracking of the tasks and milestones. It has a big advantage over other project management tools, as the revision control and the issue tracking is at the same place. Non project members can understand the thoughts behind certain decisions and communicate their ideas directly to team members.
An organization named osm2vectortiles was created with the following two repositories:

- **osm2vectortiles** contains the project[22]
- **osm2vectortiles-thesis** contains the thesis[23]

11.2 SCHEDULE

Compared to other software projects a much longer elaboration phase has been chosen. In the elaboration the most difficult problems have been solved with a prototype which was then refined in the construction phases. Due to the many risks regarding rendering time and unknown technology problems the longer elaboration phase really helped to eradicate risks early on.

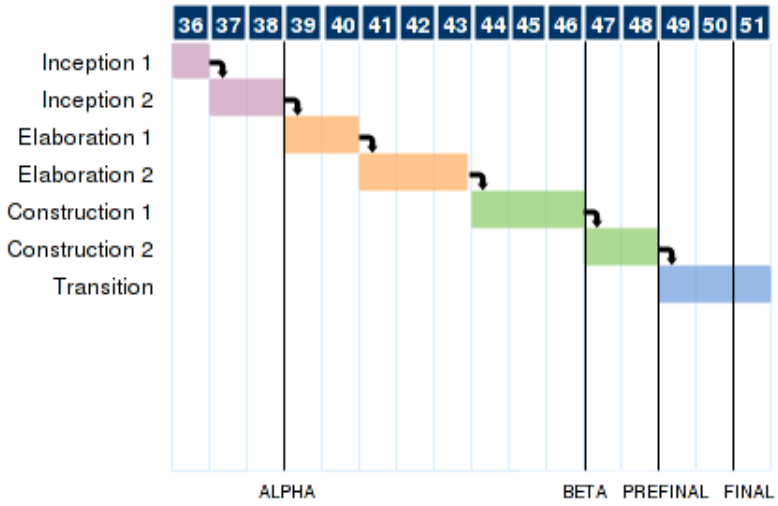

Figure 27: Phases during project

11.3 MILESTONES

Each milestones marks a special release version of the vector tiles.

ALPHA	Proof of concept tileserver.
BETA	Switzerland with upper zoom levels.
PREFINAL	Switzerland with lower zoom levels.
FINAL	Improved and polished final release.

Table 10: Project milestones

11.4 PROJECT STAGES

Inception 1	Kickoff meeting and definition of project proposal.
Inception 2	Getting familiar with the entire Mapbox stack and create a very basic prototype of every deliverable on a small scale.
Elaboration 1	Evaluation of different parts of the stack like import tools and vector tile server and project structure.
Elaboration 2	Prototype of import and export components and configuration of continuous integration.
Construction 1	Custom data style up from zoom level 8 to 14 and a good mapping configuration.
Construction 2	Custom data style up from zoom level 0 to 8 and importing external data sources.
Transition	Rendering and preparing the vector tiles and time for documentation and project website.

Table 11: Project stages

11.5 ROLES AND RESPONSIBILITIES

Prof Stefan Keller	Thesis advisor responsible for supervising work and assess the thesis.
Dr Petr Pridal	Technical partner responsible for providing infrastructure and guidance in technical and map related questions.
Manuel Roth	Contributor responsible for data style and JavaScript tooling.
Lukas Martinelli	Contributor responsible for rendering infrastructure and Python scripts.

Table 12: Thesis contributors and their roles

11.6 RISKS

In general this project was hazardous, as not a lot of cartographic knowledge was present at the beginning. The risk was reduced by contacting the advisors early, when ever problems appeared. The knowledge gap was filled very fast, due to the availability of superb information sources online. The figure below shows only project specific risks, which have been assessed and dealt with accordingly.

Risk	Measurement	Probability (1-6)
Low cartographic knowledge	Start earlier for additional training time	6
Rendering takes too long	Increase infrastructure	5
Infrastructure not sufficient	Switch to non school infrastructure and rely on external sponsors	3
No community acceptance	Meet early with OSM community	3
Quality not sufficient	Invest sufficient time in perfecting the quality	3
High technical complexity	Prototype early	6
Insufficient tracking of project progress	More frequent meetings	2

Table 13: Risks and measurements

HIGH TECHNICAL COMPLEXITY The project involved a lot of different technologies, as the workflow to produce vector tiles was built up. As a measurement the elaboration phase was used to create a first working prototype, which should prove the implementation concept.

INSUFFICIENT TRACKING OF PROJECT PROGRESS The project progress was tracked on Github, so the progress was at any time publicly visible. Regular meetings with the advisors helped to stay focused on the project goals.

INFRASTRUCTURE NOT SUFFICIENT The rendering process of vector tiles is a very ressource intense task. The school infrastructure was not enough to fulfill the project goals in time. Therefore an alternative platform with more processing power was used to complete the rendering task in time.

As this project brought up new challenges every day, not all of the risks could be eliminated during the elaboration phase. Measurements had to be found, whenever new risks appeared.

12

QUALITY MEASURES

12.1 TESTING

The osm2vectortiles ecosystem is quite diverse with a big collection of small tools that all work together.

12.2 DEBUG VIEWER

During development the tiles were continuously examined with the Mapbox Studio Classic debug viewer. This tool makes it possible to verify the different attributes of features and ensure the queries deliver the right results.

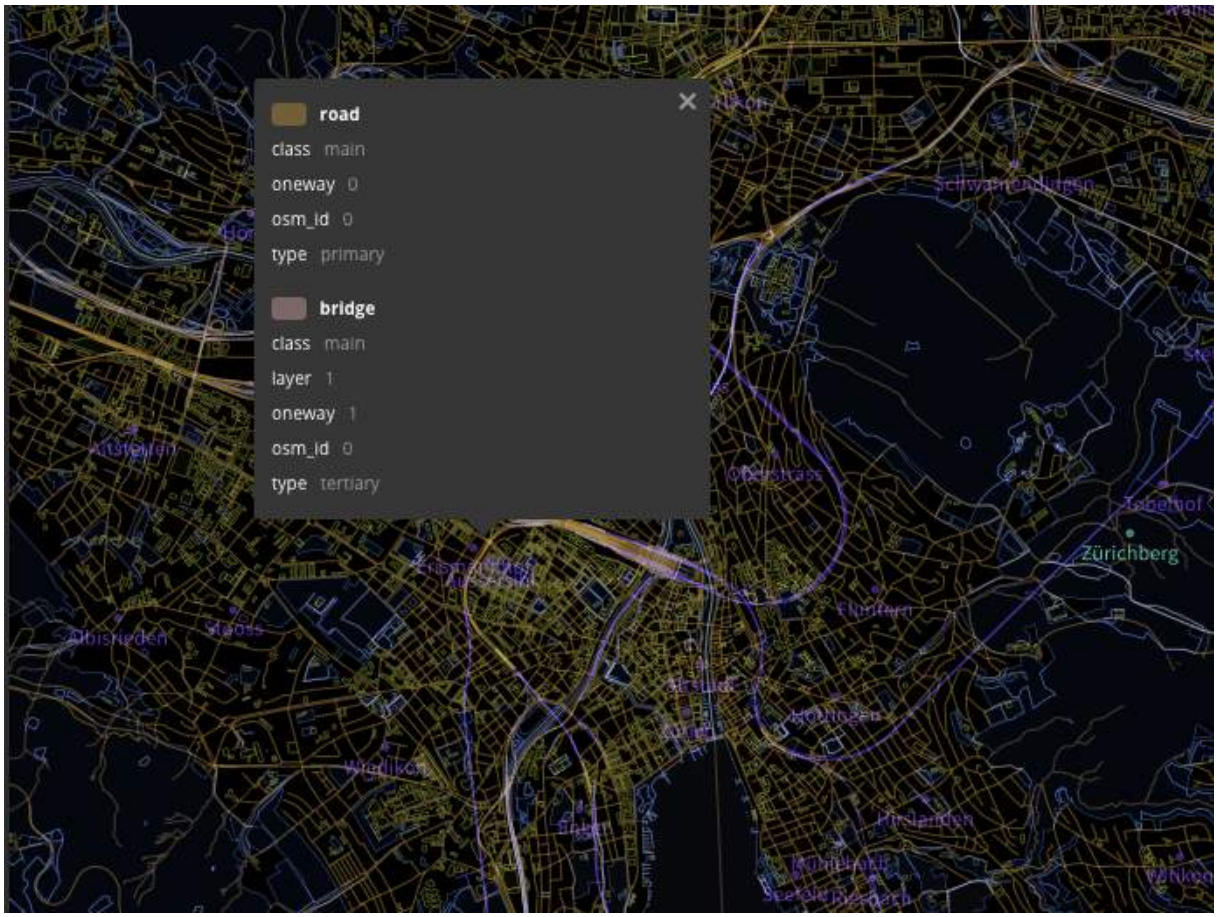

Figure 28: Mapbox Studio Classic Debug Viewer

The Open Source partner Klokan Technologies [24] also provided a OpenLayers 3[53] based debug viewer, which supports the same features but allows looking at arbitrary TileJSON urls.

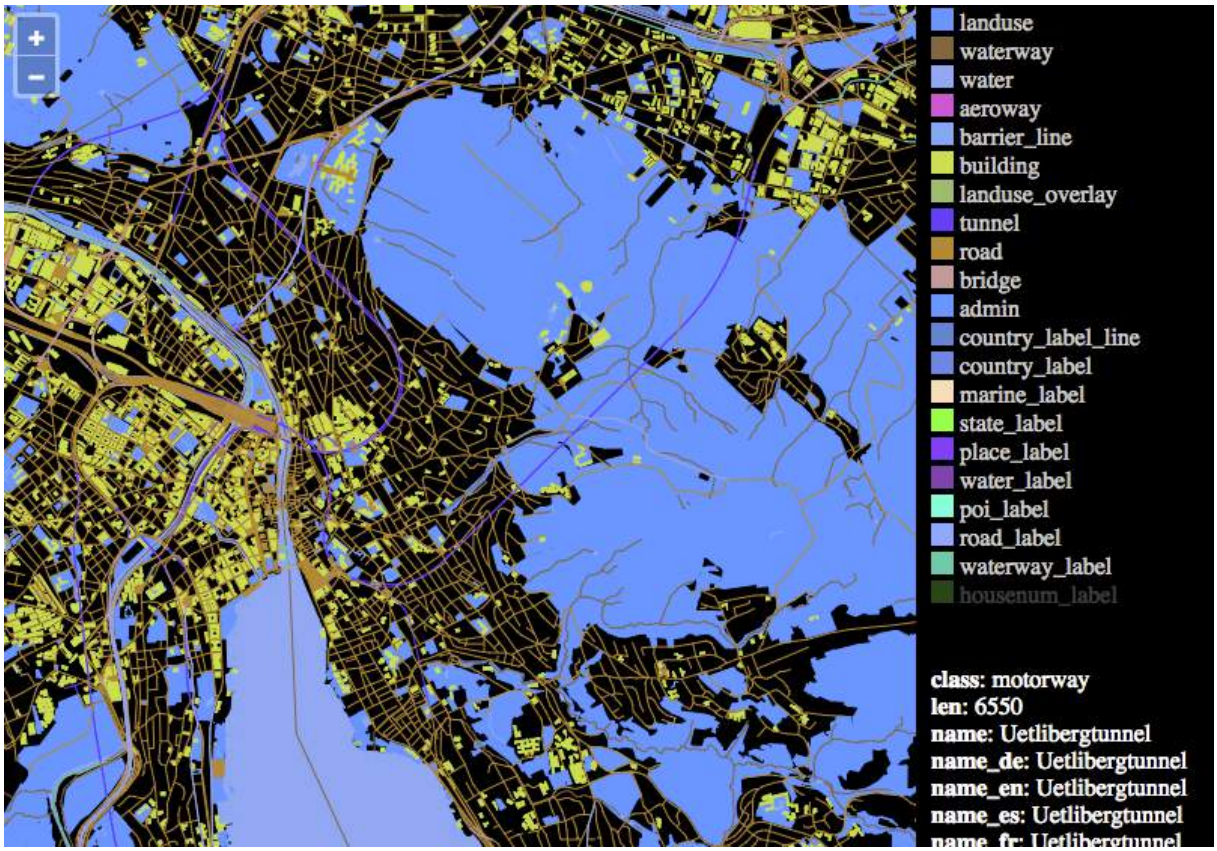

Figure 29: Klokantech Debug Viewer

The tile inspector is another tool by Klokan Technologies which supports inspecting vector tile PBFs for their size and features and visually explore the structural features of the Vector Tile Compare.

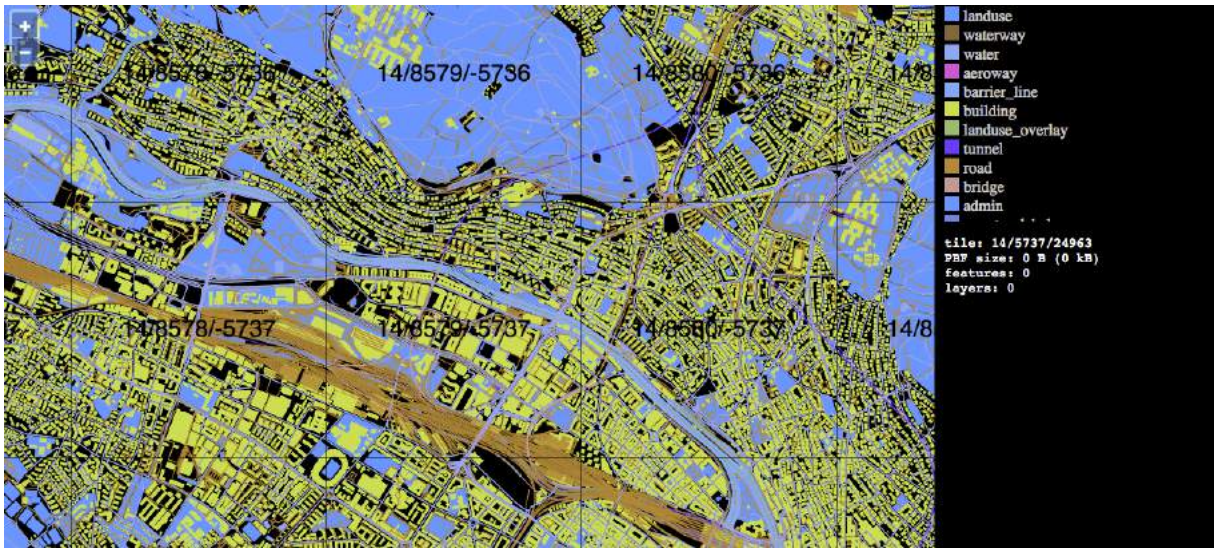

Figure 30: Klokantech Tile Inspector

12.3 VISUAL TEST

The Visual Compare tool allowed to preview the map visually and compare the rendered raster tiles directly with Mapbox Streets.

12.4 STRUCTURAL TEST

With the Vector Tile Compare tool the differences between Mapbox Streets and the Open Streets vector tiles were regularly compared by hand. This tool was used as guidance for reverse engineering and ensuring the same feature classes appear at the same zoom levels.

12.5 INTEGRATION TEST

In Travis CI[62] the entire workflow was completed for a small data sample on each commit. Because the entire workflow is configured with Docker Compose [7] the CI server had to execute all import steps in serial order. This is a straightforward way to check if all components work together correctly and although it is a simple setup it has helped tremendously during project development, catching bugs like missing tables or SQL typos.

```
script:
  - docker-compose up -d postgis
  - docker-compose up -d pgbouncer
  - docker-compose run import-osm
  - docker-compose run import-natural-earth
  - docker-compose run import-water
  - docker-compose run import-labels
  - docker-compose run import-sql
  - docker-compose run update-scaleranks
  - docker-compose run export-local
  - docker-compose up -d serve
  - curl "http://localhost:8080/index.json"
```

12.6 GUIDELINES

To have homogeneous software the contributors have settled on common guidelines in the beginning of the project.

12.6.1 *Releases*

Semantic versioning [58] should be used for releases. At the end of each milestone a new release will be created.

12.6.2 *Git*

COMMIT MESSAGES The seven rules of great git commit messages[3] should be used.

REWRITING Git history should be kept clean and therefore local branches should be squashed meaningfully.

PULLING To avoid unnecessary merge messages one should always use the `--rebase` parameter.

12.6.3 *Workflow*

The Feature Branch Workflow[63] should be used. Every project member has a local repository with a copy of the remote repository. For each feature ticket in GitHub a separate branch will be created. Once a ticket has been completed a pull request will be created and needs to be merged into the `master` branch by an other

12.6.4 *Coding Standards*

BASH Bash was used for the Docker image entrypoints and follow the rules of Defensive Bash Programming [31].

PYTHON Python code should stay PEP-8[59] compliant and write idiomatic Python code according to PEP-20[60].

JAVASCRIPT The JavaScript code is checked using ESLint[9]

SQL The PostgreSQL code is using upper case for the key words. Apart from nice formatted SQL code and functions should be used to keep the queries DRY[79].

DOCKERFILE Dockerfiles follow the best practices[6] defined by Docker.

13 PROJECT MONITORING

13.1 CODE STATISTICS

The actual Project Stages are recognizable on the GitHub code frequency chart as well. In Elaboration and Construction most commits have been done while in the Transition phase the graph flattens out.

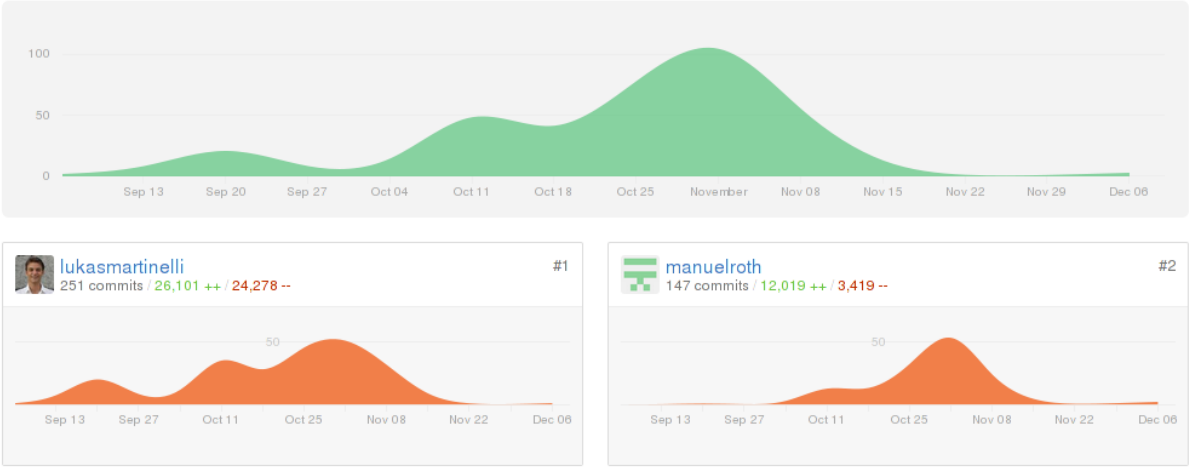

Figure 31: Commit frequency

13.2 ESTIMATED TIME VS ACTUAL TIME

Our estimations were too optimistic. Due to the extensive training period required to get started in a GIS environment the actual time was more than originally estimated.

Sprint	**Actual**	**Estimated**
Inception 1	20	16
Inception 2	42	55
Elaboration 1	48	50
Elaboration 2	121	102
Construction 1	152	106
Construction 2	48	53
Transition	61	98
Total	493	480

Table 14: Estimated vs actual time for different sprints

13.3 TIME PER PERSON

Both contributors invested about the same amount of time.

Sprint	**Lukas Martinelli**	**Manuel Roth**	**Total**
Inception 1	11	9	20
Inception 2	22	21	43
Elaboration 1	25	23	48
Elaboration 2	56	65	121
Construction 1	77	75	152
Construction 2	27	21	48
Transition	28	33	61
Total	245	248	493

Table 15: Time for each contributor for sprints

Part III

APPENDIX

A

USER DOCUMENTATION

DISPLAY RASTER MAP

This tutorial describes how to display a raster tile map with the rendered vector tiles as data source.

Preparation

1. Download[1] the appropriate extract you want to serve.
2. Download[2] a suitable visual style.
3. Add both to the same directory and make sure that the have the same name.

Install Kitematic

1. Download and install Kitematic[3].
2. Start a new container by searching for `osm2vectortiles` and click create on the container called `serve`.

1 http://osm2vectortiles.org/data/download.html
2 https://github.com/mapbox/mapbox-studio-osm-bright.tm2.git
3 https://www.docker.com/docker-toolbox

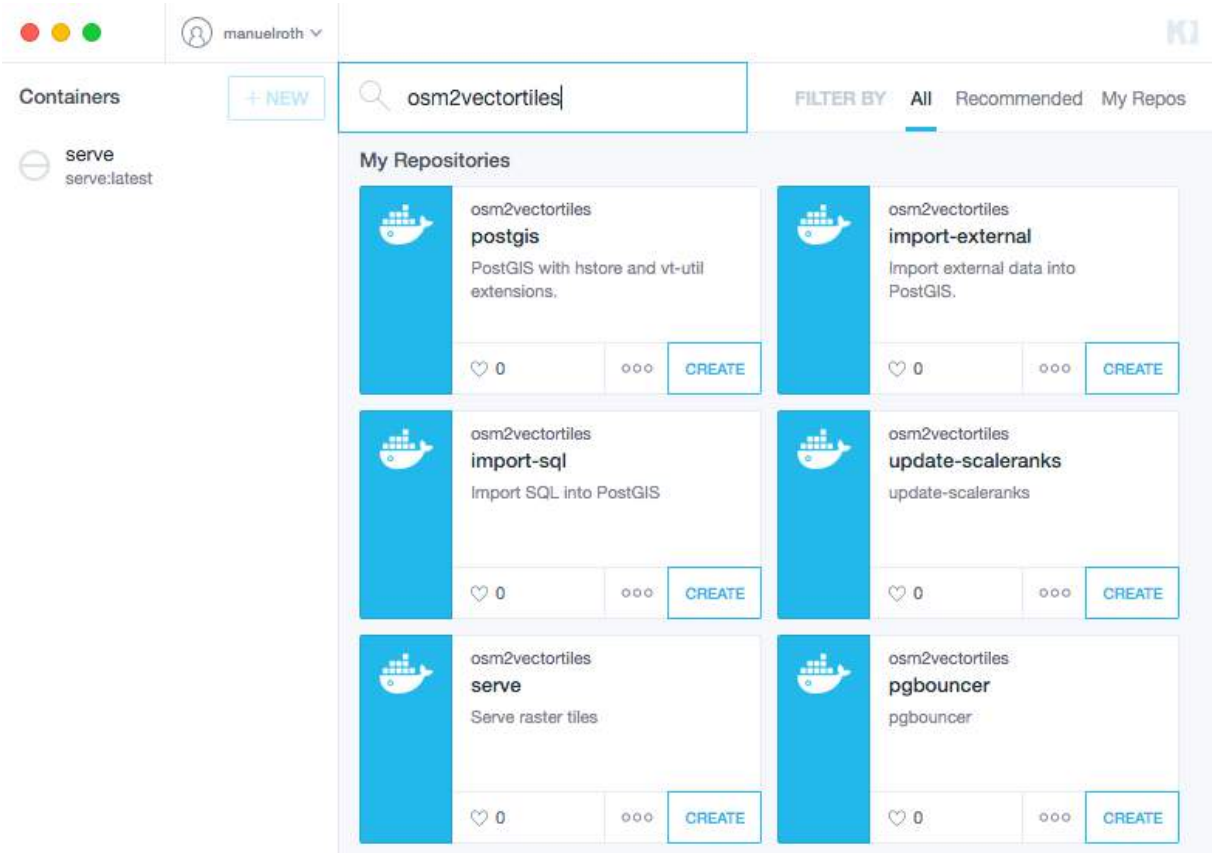

Figure 32: Search Container

Kitematic Usage

When you start the container it will complain about missing `tm2` style projects.

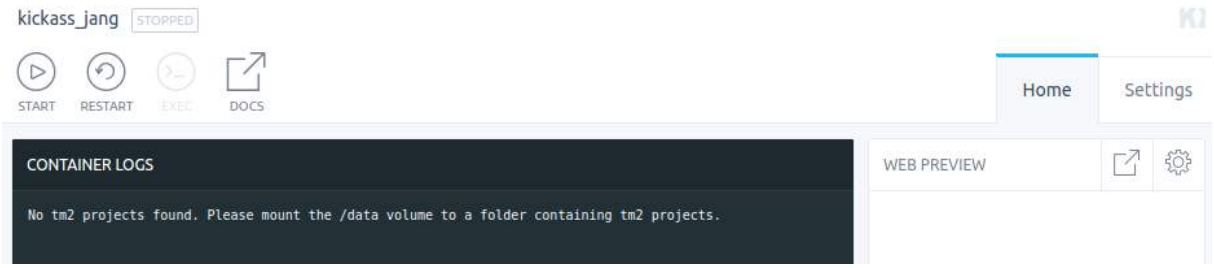

Figure 33: Container started unsucessfully

Mount your the directory containing the `mbtiles` files and `tm2` style projects into the `/data` volume.

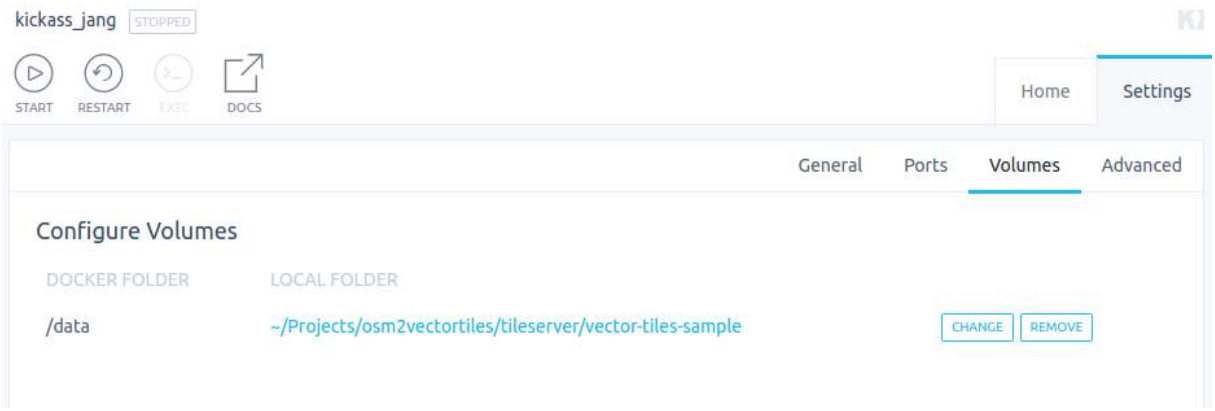

Figure 34: Configured volumes for container

Now restart the container. You should be up and running serving generated raster tiles.

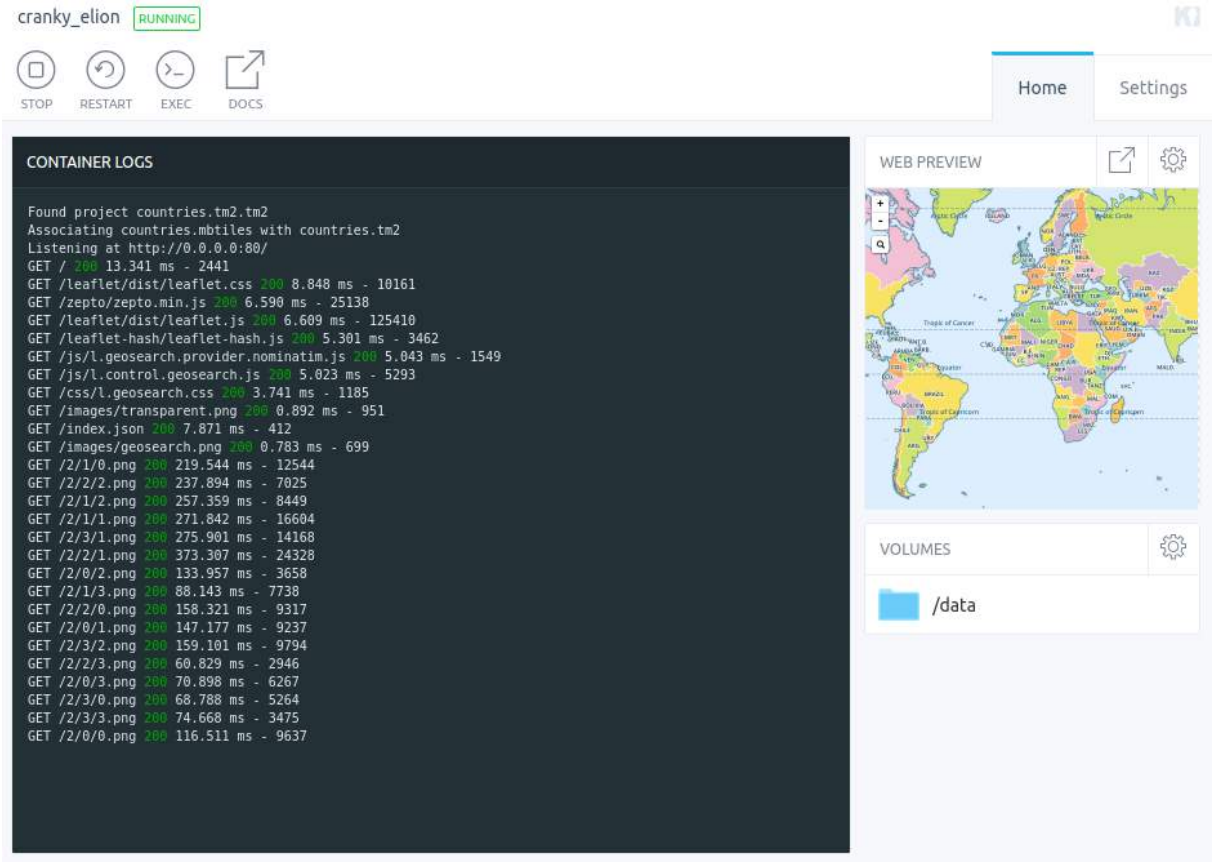

Figure 35: Container running and serving tiles

DISPLAY MAP WITH MAPBOXGL

To display a custom MapboxGL based map you need to create a HTML file and reference the public vector tile server of osm2vectortiles. You are free to download and host the vector tiles yourself but a fast and distributed CDN service for serving the PBFs is provided for you.
The easiest way to get started is using the `mapbox-gl-js-exmaple` repository. Clone the repository and change into the directory.

```
git clone https://github.com/osm2vectortiles/mapbox-gl-js-example.git
```

Configure Source, Fonts and Sprites

In order for Mapbox GL JS to work you also need to provide the font[4] and sprites[5]. These resources are contained in the folder `assets`.
The Mapbox GL Style JSON[6] of OSM Bright is located at `bright-v8.json`. You can create your own styles with Mapbox Studio.
If you want to serve the Mapbox GL Style JSON without Mapbox you need to configure three URLs.

1. Change the `sources` URL to the free osm2vectortile serve or use your own server.
2. Change the `sprite` URL to the location of your sprites.
3. Change the `glyphs` URL to the location of your fonts.

```
"sources": {
    "mapbox": {
        "url": "http://vectortiles.osm2vectortiles.org/world.json",
        "type": "vector"
    }
},
"sprite": "/assets/sprite",
"glyphs": "/assets/font/{fontstack}/{range}.pbf"
```

Initialize the Map

In order to serve a MapboxGL based map you need a Mapbox GL style JSON created with Mapbox Studio[7]

4 https://www.mapbox.com/mapbox-gl-style-spec/#glyphs
5 https://www.mapbox.com/mapbox-gl-style-spec/#sprite
6 https://www.mapbox.com/mapbox-gl-style-spec/

B

DEVELOPER DOCUMENTATION

CREATE YOUR OWN VECTOR TILES

Docker is used extensively for development and deployment. The easiest way to get started is using Docker Compose[1].

1. Step: Clone the osm2vectortiles project.

```
git clone https://github.com/osm2vectortiles/osm2vectortiles.git
```

2. Step: Start up your PostGIS container with the data container attached.

```
docker-compose up -d postgis
```

3. Step: Download a PBF and put it into the local `import` directory.

```
wget https://s3.amazonaws.com/metro-extracts.mapzen.com/zurich_switzerland.osm.pbf
```

4. Step: Now you need to import the PBF files into PostGIS.

```
docker-compose up import-osm
```

5. Step: Now you need to import several external data sources.
Import water polygons from OpenStreetMapData.

```
docker-compose up import-water
```

6. Step: Import Natural Earth data for lower zoom levels.

```
docker-compose up import-natural-earth
```

7. Step: Import custom country, sea and state labels.

```
docker-compose up import-labels
```

8. Step: Now import custom SQL functions.

1 https://www.docker.com/docker-compose

```
docker-compose up import-sql
```

9. Step: Update the scaleranks of OSM places.

```
docker-compose up update-scaleranks
```

10. Step: Export the data as MBTiles file to the `export` directory.

```
docker-compose up export
```

11. Step: Serve the tiles as raster tiles from `export` directory.

```
docker-compose up serve
```

CREATE OWN EXTRACT

If you need an extract which is not included on the downloads page[2], you can download the planet file and make your own extract.

Preparation

1. Download[3] the planet file.
2. Get the bounding box of your desired extract.[4]
3. Install tilelive utility.

   ```
   npm install -g tilelive
   ```

Create Extract

To create an extract the `tilelive-copy` utility is used. It takes a bounding box and a MBTiles file as input and ouputs the extract.
Replace the bounding box in the following command with your bounding box.

```
tilelive-copy --minzoom=0 --maxzoom=14 --bounds="60.403889,29.288333,74.989862,38.5899217"
world.mbtiles switzerland.mbtiles
```

2 http://osm2vectortiles.org/data/download.html
3 http://osm2vectortiles.org/data/download.html
4 http://tools.geofabrik.de/calc/#type=geofabrik_standard&bbox=5.538062,47.236312,15.371071,54.954937&tab=1&proj=EPSG:4326&places=2

LAYER REFERENCE

This is a guide to the data inside the OSM Vector Tiles to help you with styling.
Available layers:

- landuse
- waterway
- water
- aeroway
- barrier_line
- building
- landuse_overlay
- tunnel
- road
- bridge
- admin
- country_label
- marine_label
- place_label
- water_label
- poi_label
- road_label
- waterway_label
- housenum_label

landuse

This layer includes polygons representing both land-use and land-cover.
It's common for many different types of landuse/landcover to be overlapping, so the polygons in this layer are ordered by the area of their geometries to ensure smaller objects will not be obscured by larger ones. Pay attention to use of transparency when styling - the overlapping shapes can cause muddied or unexpected colors.

water

This is a simple polygon layer with no differentiating types or classes. Water bodies are filtered and simplified according to scale - only oceans and very large lakes are shown at the lowest zoom levels, while smaller and smaller lakes and ponds appear as you zoom in.

waterway

The waterway layer contains rivers, streams, canals, etc represented as lines.

Type

The `type` column can contain one of the following types:

- `ditch`
- `stream`
- `stream_intermittent`
- `river`
- `canal`
- `drain`
- `ditch`

aeroway

The aeroway layer contains the types runway, taxiway, apron and helipad.

barrier_line

The barrier_line layer contains the classes fence, cliff, gate.

building

This layer contains buildings. Buildings are shown starting zoom level 13.

landuse_overlay

This layer is for landuse polygons that should be drawn above the #water layer.

tunnel, road, bridge

The layers tunnel and bridge are based of the layer road.

Class

The main field used for styling the tunnel, road, and bridge layers is the `class` field.

Class	Aggregated Types
motorway	motorway
motorway_link	motorway_link
driveway	driveway
main	primary, primary_link, trunk, trunk_link, secondary, secondary_link, tertiary, tertiary_link
street	residential, unclassified, living_street, road, raceway
street_limited	pedestrian, construction, private
service	service, track, alley. spur, siding, crossover
path	path, cycleway, ski, steps, bridleway, footway
major_rail	rail, monorail, narrow_gauge, subway
aerialway	chair_lift, mixed_lift, drag_lift, platter, t-bar, magic_carpet, gondola, cable_car, rope_tow, zip_line, j-bar, canopy
golf	hole

admin

This layer contains the administrative boundary lines. These are based on Natural Earth data on lower zoom levels (0-6) and OSM data (7-14) on upper zoom levels.

Administrative Levels

The administrative levels are in the `admin` field.

Value	Aggregated Types
2	countries
4	states, provinces

The `disputed` field should be used to apply a dashed or otherwise distinct style to disputed boundaries.

Maritime Boundaries

The `maritime` field can be used as a filter to downplay or hide maritime boundaries, which are often not shown on maps.

Localrank

The `country_label` layer contains the labels of all countries with translated names.

Localrank

The `scalerank` field is used to hide or show the label based on the importance, size and available room.

marine_label

The marine_label layer contains labels for marine features such as oceans, seas, large lakes and bays. The `labelrank` is used to hide or show the label based on the importance, size and available room.

state_label

The layer state_label contains labels for large provinces in large countries such as China, USA, Russia, Australia and UK.

place_label

The layer place_label contains labels for cities.

Scalerank

The scalerank is used to hide or show the label based on the importance, size and available room.

Localrank

The `localrank` field can be used to adjust the label density by showing fewer labels.

water_label

The layer water_label contains labels for bodies of water such as lakes.

road_label

The layer road_label uses the same classification as the layer road.

waterway_label

The layer waterway_label contains labels for rivers.

housenum_label

This layer contains points used to label the street number parts of specific addresses. Both housenumber polygons and points were mapped to a single layer. The `house_num` field countains house and building numbers.

poi_label

The poi_label layer is used to place icons and labels for various point of interests.

Names

Names are available in all languages (name, name_en, name_de, name_fr, name_es, name_ru, name_zh).

Scalerank

The scalerank of a POI is determined by the area.

Scalerank	Description
1	The POI has a very large area >145000
2	The POI has a medium-large area >12780
3	The POI has a small area 2960 or is a station
4	The POI has no known area

Localrank

The localrank field can be used to adjust the label density by showing fewer labels. The localrank is a whole number which starts at 1 and groups places in a grid by their importance. Importance of POIs are weighted in the following order:

1. station, subway_entrance, park, cemetery, bank, supermarket, car, library, university, college, police, townhall, courthouse
2. nature_reserve, garden, public_building
3. stadium
4. hospital
5. zoo
6. university, school, college, kindergarten
7. supermarket, department_store
8. nature_reserve, swimming_area
9. attraction

Maki Labels and Types

The `type` field in the poi_label layer is mapped to the appropriate Maki label which can be queried in maki. Types are stored in a human readable format in the data where chair_lift becomes Chair lift so you can use the type field for as label.

BIBLIOGRAPHY

[1] Axismaps.github.io. Labeling and text hierarchy in cartography, 2015. URL https://axismaps.github.io/thematic-cartography/articles/labeling.html. Visited on 2015-12-14.

[2] Josh Berkus, Josh Berkus, and View profile. Database soup: Running with scissors mode, 2015. URL http://www.databasesoup.com/2015/02/running-with-scissors-mode.html. Visited on 2015-12-14.

[3] Chris.beams.io. How to write a git commit message, 2015. URL http://chris.beams.io/posts/git-commit/. Visited on 2015-12-14.

[4] Dave Cole. Mapbox streets: A global map with street level detail, 2015. URL https://www.mapbox.com/blog/announcing-mapbox-streets/. Visited on 2015-12-14.

[5] Container42.com. Persistent volumes with docker - data-only container pattern · container42, 2015. URL http://container42.com/2013/12/16/persistent-volumes-with-docker-container-as-volume-pattern/. Visited on 2015-12-14.

[6] Docs.docker.com. Best practices for writing dockerfiles, 2015. URL https://docs.docker.com/engine/articles/dockerfile_best-practices/. Visited on 2015-12-14.

[7] Docs.docker.com. Docker compose, 2015. URL https://docs.docker.com/compose/. Visited on 2015-12-14.

[8] Download.geofabrik.de, 2015. URL http://download.geofabrik.de/. Visited on 2015-12-14.

[9] Eslint.org. Eslint - pluggable javascript linter, 2015. URL http://eslint.org/. Visited on 2015-12-14.

[10] Node.js Foundation. Node.js, 2015. URL https://nodejs.org/en/. Visited on 2015-12-14.

[11] Julien Gaffuri. Toward web mapping with vector data. In *Geographic Information Science*, pages 87–101. Springer, 2012.

[12] Gdal.org. Gdal: ogr2ogr, 2015. URL http://www.gdal.org/ogr2ogr.html. Visited on 2015-12-14.

[13] GitHub. Mapbox carto css, 2014. URL https://github.com/mapbox/carto/blob/master/docs/latest.md. Visited on 2015-12-14.

[14] GitHub. Awesome vector tiles, 2015. URL https://github.com/mapbox/awesome-vector-tiles. Visited on 2015-12-14.

[15] GitHub. Tessera vector tile server, 2015. URL https://github.com/mojodna/tessera. Visited on 2015-12-14.

[16] GitHub. Mapbox tilelive, 2015. URL https://github.com/mapbox/tilelive. Visited on 2015-12-14.

[17] GitHub. Tilemaker, 2015. URL https://github.com/systemed/tilemaker. Visited on 2015-12-14.

[18] GitHub. Vector tile specification, 2015. URL https://github.com/mapbox/vector-tile-spec/tree/master/1.0.1. Visited on 2015-12-14.

[19] GitHub. A command line alternative to tilelive, 2015. URL https://github.com/mojodna/tl. Visited on 2015-12-14.

[20] GitHub. Osm2pgsql default schema, 2015. URL https://github.com/openstreetmap/osm2pgsql/blob/master/default.style. Visited on 2015-12-14.

[21] GitHub. Mapbox osm bright visual style, 2015. URL https://github.com/mapbox/mapbox-studio-osm-bright.tm2. Visited on 2015-12-14.

[22] GitHub. Osm2vectortiles project, 2015. URL https://github.com/osm2vectortiles/osm2vectortiles. Visited on 2015-12-14.

[23] GitHub. Osm2vectortiles project thesis, 2015. URL https://github.com/osm2vectortiles/osm2vectortiles-thesis. Visited on 2015-12-14.

[24] Klokan GmbH. Klokan technologies, 2015. URL http://www.klokantech.com/. Visited on 2015-12-14.

[25] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[26] Imposm.org. Imposm3 diff import, 2015. URL http://imposm.org/docs/imposm3/latest/tutorial.html#diff. Visited on 2015-12-14.

[27] Imposm.org. Imposm, 2015. URL http://imposm.org/. Visited on 2015-12-14.

[28] Imposm.org. Data mapping — imposm3 3.0.0a documentation, 2015. URL http://imposm.org/docs/imposm3/latest/mapping.html. Visited on 2015-12-14.

[29] Imposm.org. Database schema, 2015. URL http://imposm.org/docs/imposm/latest/database_schema.html. Visited on 2015-12-14.

[30] Kitematic. Kitematic, 2015. URL https://kitematic.com/. Visited on 2015-12-14.

[31] Kfir Lavi. Defensive bash programming, 2012. URL http://www.kfirlavi.com/blog/2012/11/14/defensive-bash-programming/. Visited on 2015-12-14.

[32] Leafletjs.com. Leaflet, 2015. URL http://leafletjs.com/. Visited on 2015-12-14.

[33] Mapbox. Atlas server, 2015. URL https://www.mapbox.com/atlas/. Visited on 2015-12-14.

[34] Mapbox.com. Maki | a clean point of interest icon set from mapbox | mapbox, 2015. URL https://www.mapbox.com/maki/. Visited on 2015-12-14.

[35] Mapbox.com. Mapbox terrain | mapbox, 2015. URL https://www.mapbox.com/developers/vector-tiles/mapbox-terrain/. Visited on 2015-12-14.

[36] Mapbox.com. Mapbox streets v6 | mapbox, 2015. URL https://www.mapbox.com/developers/vector-tiles/mapbox-streets-v6/. Visited on 2015-12-14.

[37] Mapbox.com. Mapbox streets v5 | mapbox, 2015. URL https://www.mapbox.com/developers/vector-tiles/mapbox-streets-v5/. Visited on 2015-12-14.

[38] Mapbox.com. Vector tiles | mapbox, 2015. URL https://www.mapbox.com/developers/vector-tiles/. Visited on 2015-12-14.

[39] Mapbox.com. Mapbox studio classic source manual | mapbox, 2015. URL https://www.mapbox.com/help/source-manual/#buffers. Visited on 2015-12-14.

[40] Mapbox.com. Mapbox studio classic source manual | mapbox, 2015. URL https://www.mapbox.com/help/source-manual/#overzooming. Visited on 2015-12-14.

[41] Mapbox.com. Mapbox | design and publish beautiful maps, 2015. URL https://www.mapbox.com/mapbox-studio-classic/. Visited on 2015-12-14.

[42] Mapbox.com. Mapbox gl, 2015. URL https://www.mapbox.com/mapbox-gl-js/api/. Visited on 2015-12-14.

[43] Mapzen.com. Mapzen vector tile service, 2015. URL https://mapzen.com/projects/vector-tiles/. Visited on 2015-12-14.

[44] Mapzen.com. Mapzen about, 2015. URL https://mapzen.com/about/. Visited on 2015-12-14.

[45] Mediawiki.org. Wikimedia maps, 2015. URL https://www.mediawiki.org/wiki/Maps. Visited on 2015-12-14.

[46] Mike.teczno.com. the liberty of postgreslessness: tiled vectors in mapnik (tecznotes), 2013. URL http://mike.teczno.com/notes/postgreslessness-mapnik-vectiles.html. Visited on 2015-12-14.

[47] Naturalearthdata.com. Natural earth, 2015. URL http://www.naturalearthdata.com/. Visited on 2015-12-14.

[48] Naturalearthdata.com. Populated places | natural earth, 2015. URL http://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-populated-places/. Visited on 2015-12-14.

[49] Naturalearthdata.com. Lakes + reservoirs | natural earth, 2015. URL http://www.naturalearthdata.com/downloads/10m-physical-vectors/10m-lakes/. Visited on 2015-12-14.

[50] Naturalearthdata.com. Admin 0 – countries | natural earth, 2015. URL http://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-0-countries/. Visited on 2015-12-14.

[51] Naturalearthdata.com. Admin 1 – states, provinces | natural earth, 2015. URL http://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-1-states-provinces/. Visited on 2015-12-14.

[52] Naturalearthdata.com. Admin 0 – breakaway, disputed areas | natural earth, 2015. URL http://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-0-breakaway-disputed-areas/. Visited on 2015-12-14.

[53] Openlayers.org. Openlayers 3, 2015. URL http://openlayers.org/. Visited on 2015-12-14.

[54] Openstreetmapdata.com. Openstreetmapdata, 2015. URL http://openstreetmapdata.com/. Visited on 2015-12-14.

[55] Openstreetmapdata.com. Water polygons | data | openstreetmapdata, 2015. URL http://openstreetmapdata.com/data/water-polygons. Visited on 2015-12-14.

[56] Pgtune.leopard.in.ua. Pgtune, 2015. URL http://pgtune.leopard.in.ua/. Visited on 2015-12-14.

[57] Postgresql.org. Postgresql: Documentation: 9.3: Introduction, 2015. URL http://www.postgresql.org/docs/9.3/static/gist-intro.html. Visited on 2015-12-14.

[58] Tom Preston-Werner. Semantic versioning, 2015. URL http://semver.org/. Visited on 2015-12-14.

[59] Python.org. Python pep 8, 2015. URL https://www.python.org/dev/peps/pep-0008/. Visited on 2015-12-14.

[60] Python.org. Python pep 20, 2015. URL https://www.python.org/dev/peps/pep-0020/. Visited on 2015-12-14.

[61] Dane Springmeyer. Mapbox vector tile specification adopted by esri, 2015. URL https://www.mapbox.com/blog/vector-tile-adoption/. Visited on 2015-12-14.

[62] Travis-ci.org. Travis ci, 2015. URL http://travis-ci.org. Visited on 2015-12-14.

[63] Atlassian Git Tutorial. Feature branch workflow, 2015. URL https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow/. Visited on 2015-12-14.

[64] Wikimediafoundation.org. Wikimedia foundation, 2015. URL https://wikimediafoundation.org/wiki/. Visited on 2015-12-14.

[65] Wiki.openstreetmap.org. Osm xml - openstreetmap wiki, 2015. URL https://wiki.openstreetmap.org/wiki/OSM_XML. Visited on 2015-12-14.

[66] Wiki.openstreetmap.org. Tags - openstreetmap wiki, 2015. URL http://wiki.openstreetmap.org/wiki/Tags. Visited on 2015-12-14.

[67] Wiki.openstreetmap.org. Overpass api - openstreetmap wiki, 2015. URL https://wiki.openstreetmap.org/wiki/Overpass_API. Visited on 2015-12-14.

[68] Wiki.openstreetmap.org. mod_tile, 2015. URL http://wiki.openstreetmap.org/wiki/Mod_tile. Visited on 2015-12-14.

[69] Wiki.openstreetmap.org. Databases and data access apis - openstreetmap wiki, 2015. URL http://wiki.openstreetmap.org/wiki/Databases_and_data_access_APIs#Database_Schemas. Visited on 2015-12-14.

[70] Wiki.openstreetmap.org. Osm2pgsql - openstreetmap wiki, 2015. URL http://wiki.openstreetmap.org/wiki/Osm2pgsql. Visited on 2015-12-14.

[71] wiki.openstreetmap.org. Osm planet diffs, 2015. URL http://wiki.openstreetmap.org/wiki/Planet.osm/diffs. Visited on 2015-12-14.

[72] Wiki.openstreetmap.org. Osm2pgsql schema, 2015. URL http://wiki.openstreetmap.org/wiki/Osm2pgsql/schema. Visited on 2015-12-14.

[73] Wiki.openstreetmap.org. Osm2pgsql import style, 2015. URL http://wiki.openstreetmap.org/wiki/Osm2pgsql#Import_style. Visited on 2015-12-14.

[74] Wiki.openstreetmap.org. 64-bit identifiers - openstreetmap wiki, 2015. URL https://wiki.openstreetmap.org/wiki/64-bit_Identifiers. Visited on 2015-12-14.

[75] Wiki.openstreetmap.org. Key:name - openstreetmap wiki, 2015. URL https://wiki.openstreetmap.org/wiki/Key:name. Visited on 2015-12-14.

[76] Wiki.openstreetmap.org. Tagfinder - openstreetmap wiki, 2015. URL https://wiki.openstreetmap.org/wiki/TagFinder. Visited on 2015-12-14.

[77] Wiki.openstreetmap.org. Pbf format - openstreetmap wiki, 2015. URL http://wiki.openstreetmap.org/wiki/PBF_Format. Visited on 2015-12-14.

[78] Wiki.openstreetmap.org. Slippy map, 2015. URL http://wiki.openstreetmap.org/wiki/Slippy_Map. Visited on 2015-12-14.

[79] Wikipedia. Don't repeat yourself, 2015. URL https://en.wikipedia.org/w/index.php?title=Don%27t_repeat_yourself&oldid=691047461. Visited on 2015-12-14.

[80] Wikipedia. Apple maps — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Apple_Maps&oldid=691885907. Visited on 2015-11-25.

[81] Wikipedia. Google maps — wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/w/index.php?title=Google_Maps&oldid=692381008. Visited on 2015-11-25.

[82] Wiki.postgresql.org. Tuning your postgresql server - postgresql wiki, 2015. URL https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server. Visited on 2015-12-14.

DECLARATION

Hereby we acknowledge,

- that we conducted this thesis by ourselves and without any external help, except with those, which are explicitly mentioned,
- that all used sources are cited academically correct, and
- that I didn't use any copyright protected materials (e.g. images) in an unauthorized manner.

Rapperswil, Fall 2015

Lukas Martinelli, June 16, 2016

Manuel Roth, June 16, 2016