MapnikTileServer

Release 0.1.0

OHDM Team

Jun 08, 2020

CONTENTS:

CHAPTER

ONE

INTRO

The MapntikTileServer is a time sensitive mapnik based tile server for OpenStreetMap and Open Historical Data Map. Written in python with the web framework Django and the official OpenStreetMap style sheets.

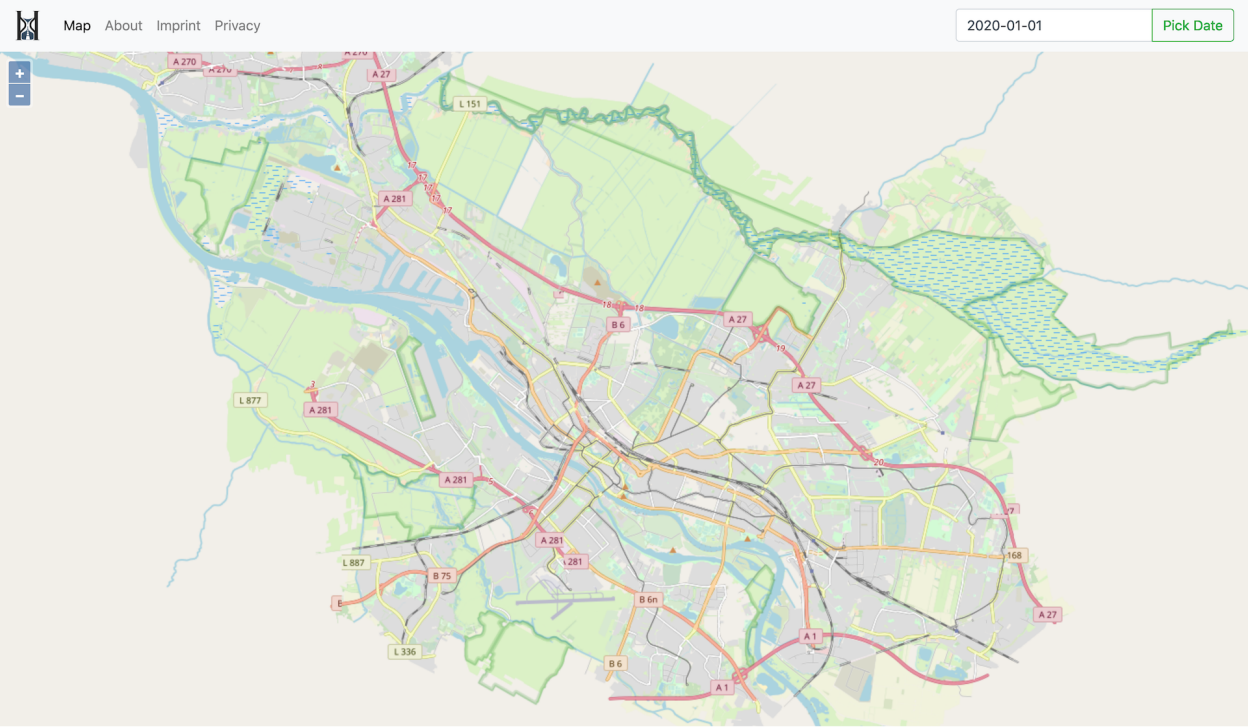

Fig. 1.1: MapntikTileServer Frontend

1.1 What is a Tile?

A tile is a part of a map. On each zoom level, the map is split into zoom^4 map parts (tiles).

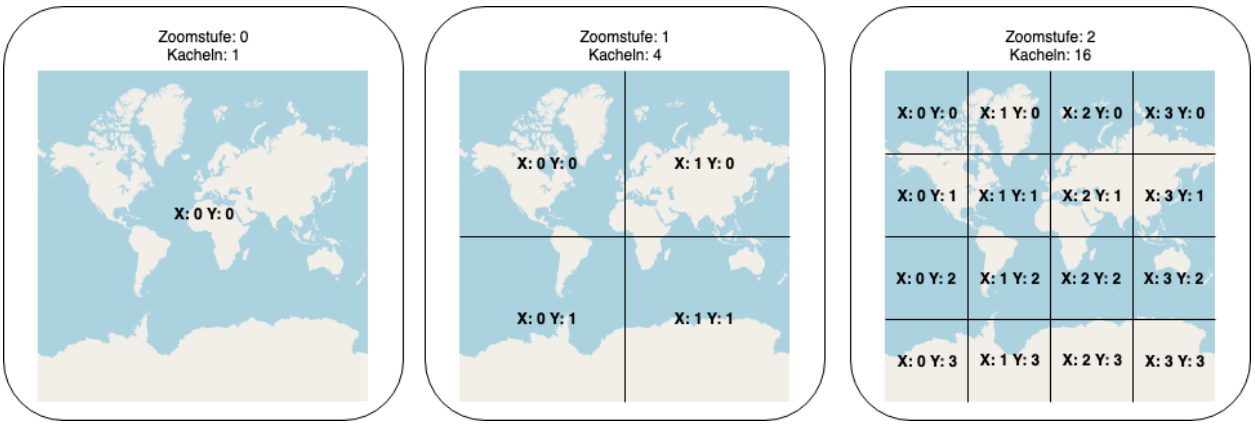

Fig. 1.2: Tile

1.2 What is a Tile server?

A Tile Server is a web service which handle user request over the `HTTP` / `HTTPS` protocol. A user request is defined over the request URL, as response get the user a single tile of a map.

There are some JavaScript libraries, which can handle tile server request and merge each tile to a map. The two most common libraries are Leaflet and OpenLayers.

1.3 MapntikTileServer vs other Tile-Servers

The main difference is, that this tile server can handle time request. So you can request a map of a specific date. To handle date specific request, the project openstreetmap-carto was use as base. Openstreetmap-carto is the github repository for the style sheets, which are used on https://openstreetmap.org/. To add time sensitive request, the project was forked and the `project.mml` file was modified to make `SQL` request for a specific day.

On the diagram below, is the workflow of the MapntikTileServer.

1.4 Cookiecutter Django

The project was created with Cookiecutter Django and build up with docker. So if you unsure how to use this MapntikTileServer, read the Cookiecutter Django Docs for help.

For faster developing and better testing, this project use Docker and Docker-Compose to build and run the MapntikTileServer.

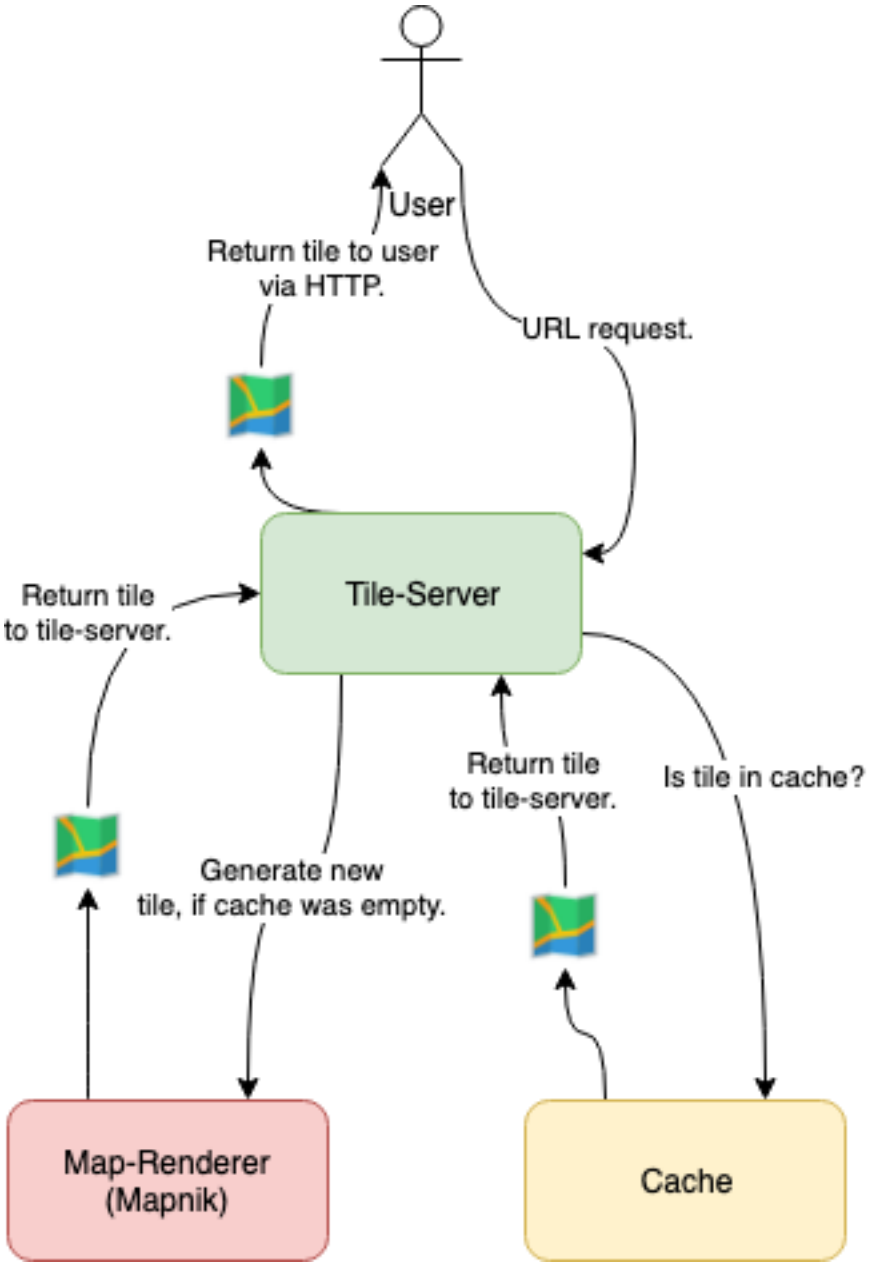

Fig. 1.3: Tile Server overview

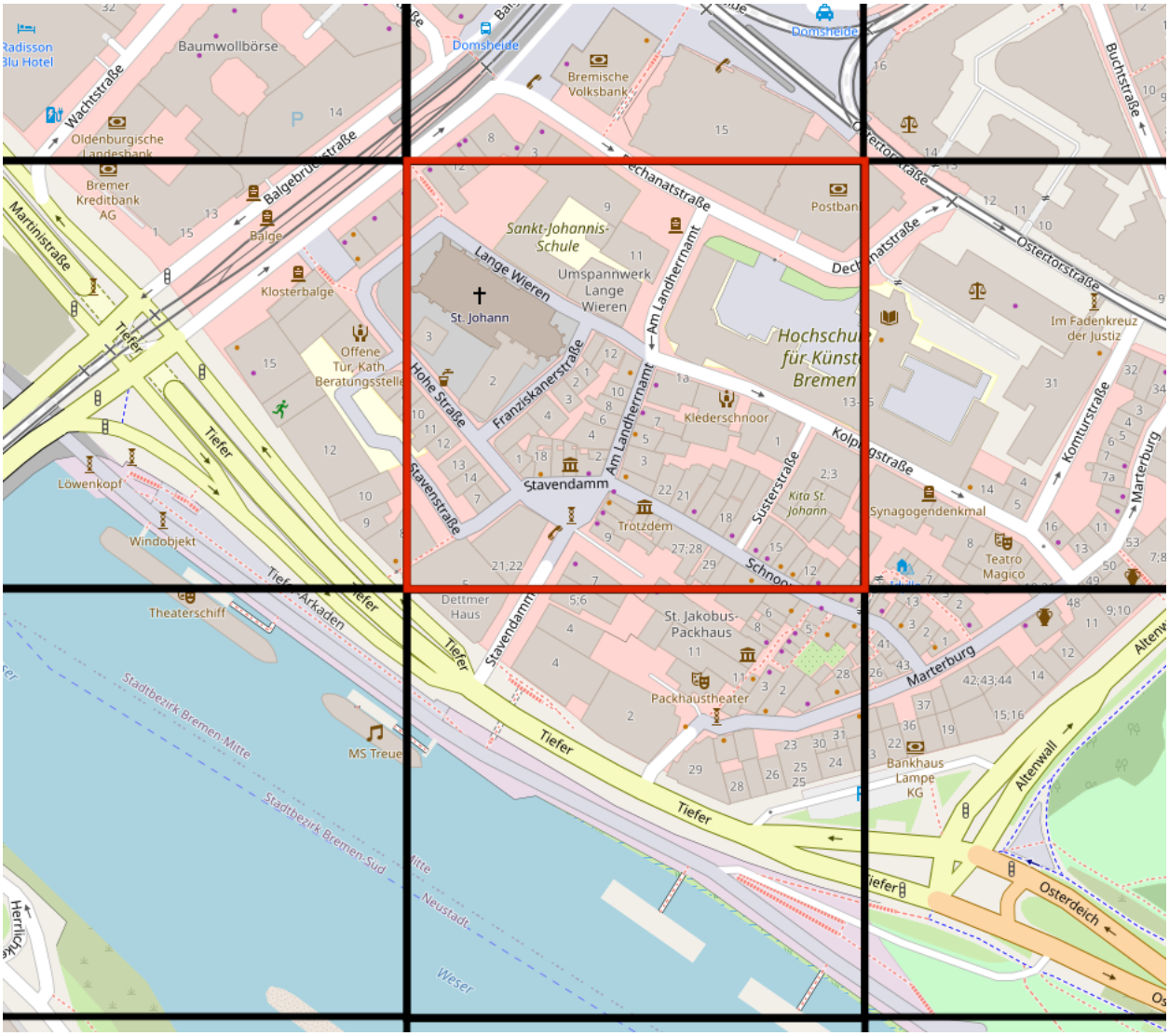

Fig. 1.4: tiles of a map

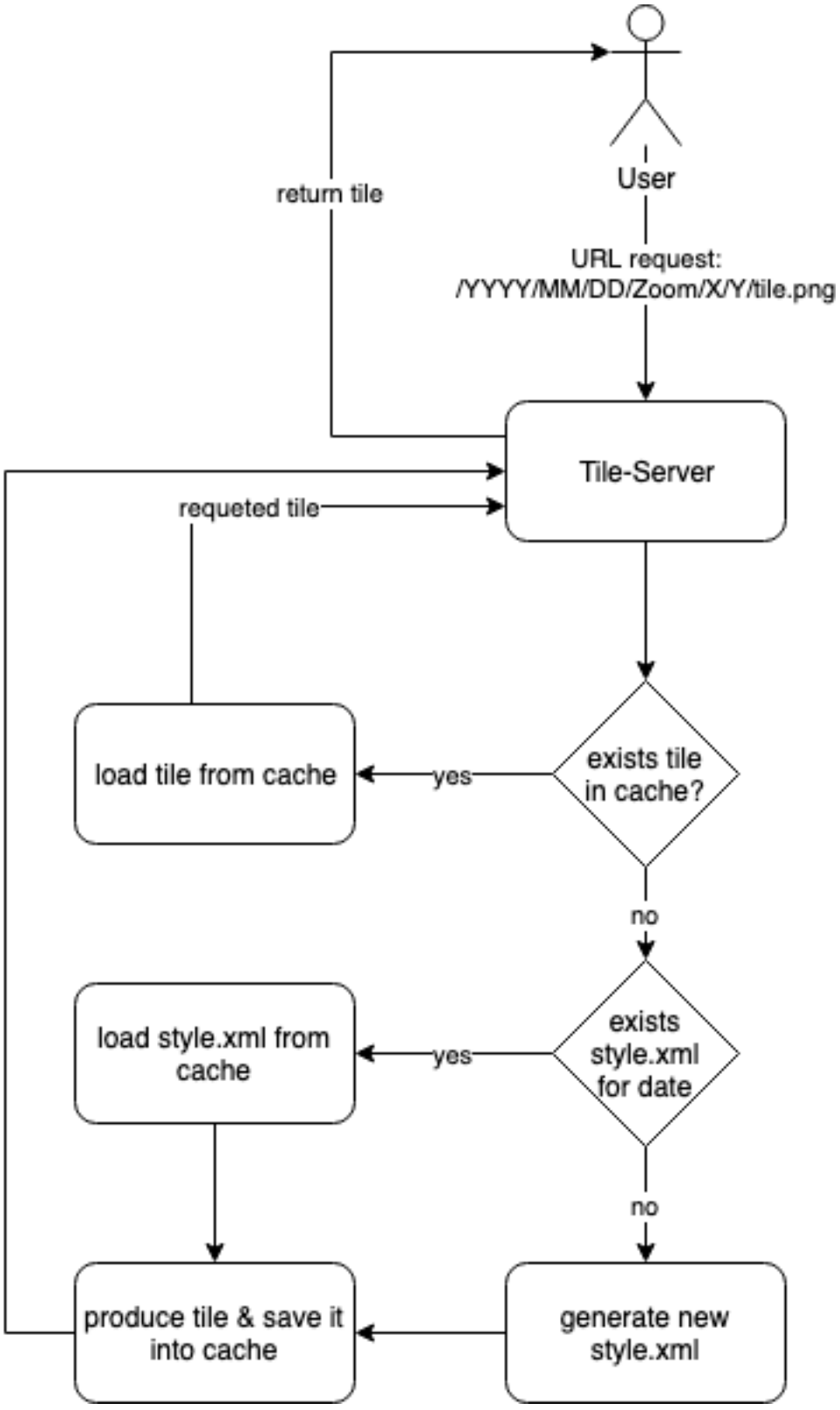

Fig. 1.5: MapntikTileServer workflow

1.5 Celery Task Queue

For the production setup, the tile are produce in a Celery-Task-Queue. Celery is a python task queue, which run in extra threads / container for more performance. For every URL of a tile, which is not already in the cache, there will be triggered a new task on the queue.

Fig. 1.6: Celery-worker get work from broker.

Every celery container have multiple threads, where it can process tile request. To scale up the production you can use with docker:

```
$ docker-compose -f production.yml scale celeryworker=2
```

Celery auto scale the threads, depending on your system load, as a recommendation is to scale celery to 2 containers.

More on: https://cookiecutter-django.readthedocs.io/en/latest/deployment-with-docker.html#building-running-production-stack

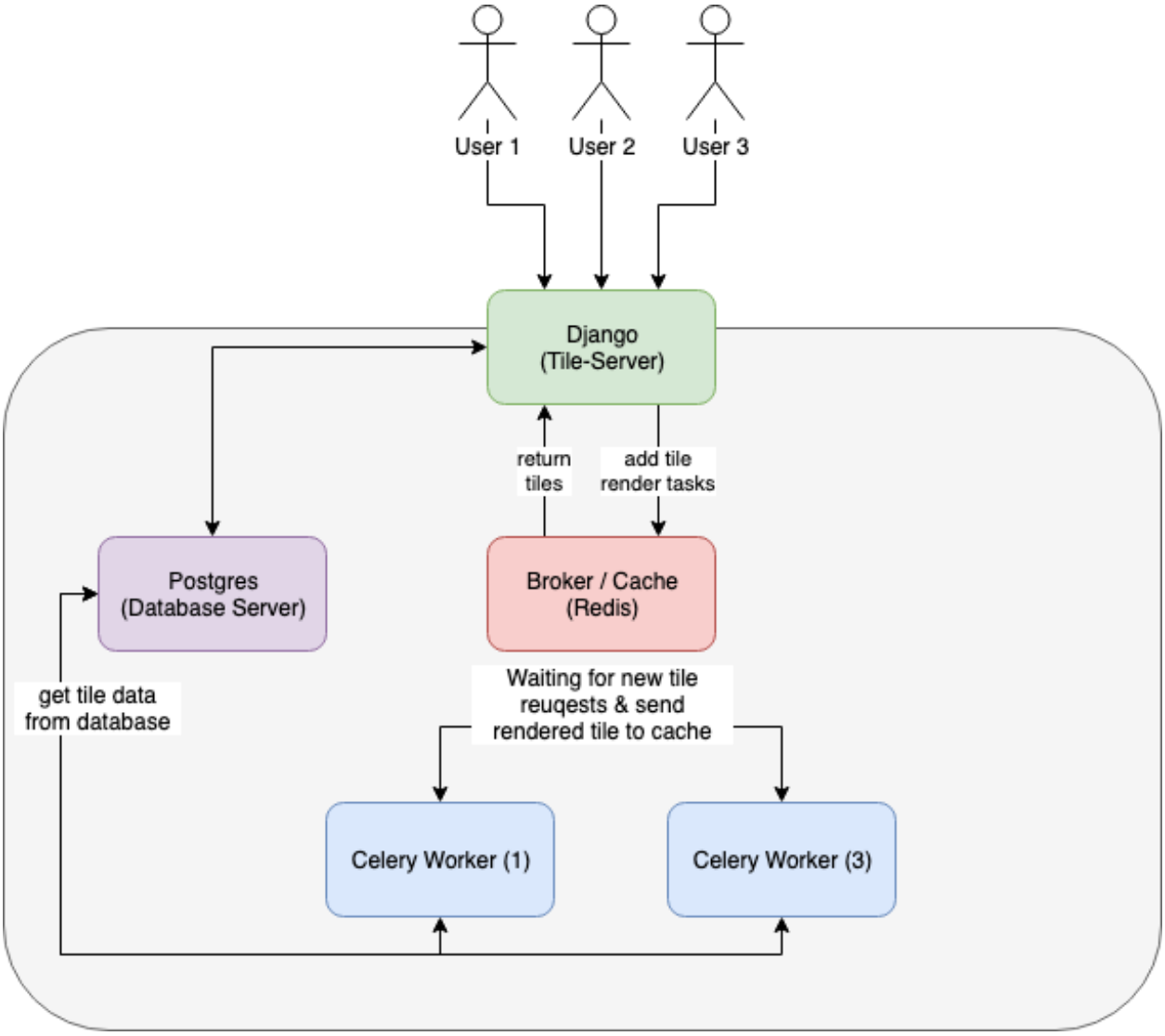

Fig. 1.7: Celery-Task-Queue

CHAPTER

TWO

GETTING UP AND RUNNING LOCALLY WITH DOCKER

The steps below will get you up and running with a local development environment. All of these commands assume you are in the root of your generated project.

2.1 Prerequisites

- Docker; if you don't have it yet, follow the installation instructions;
- Docker Compose; refer to the official documentation for the installation guide.

2.2 Install

Download the MapnikTileServer & openstreetmap-carto.:

```
$ git clone git@github.com:OpenHistoricalDataMap/MapnikTileServer.git
$ git clone git@github.com:linuxluigi/openstreetmap-carto.git
```

Go to the MapnikTileServer and build up all images. This could take some time.:

```
$ cd MapnikTileServer
$ docker-compose -f local.yml build
```

Download the shape files.:

```
$ docker-compose -f local.yml run --rm django /get-shapefiles.sh
```

Create the base mapnik style XML.:

```
$ docker-compose -f local.yml run --rm django python manage.py create_style_xml
```

Start the postgres server and create the mapnik tables.:

```
$ docker-compose -f local.yml up -d postgres
$ docker-compose -f local.yml run --rm django python manage.py migrate
```

Create a demo OHDM database.:

```
$ docker-compose -f local.yml up test-database
```

Convert the OHDM data into mapnik tables (osm2pgsql).:

```
$ docker-compose -f local.yml run --rm django python manage.py ohdm2mapnik
```

Start celery worker & beat.:

```
$ docker-compose -f local.yml up -d celerybeat celeryworker
```

Start the django web server.:

```
$ docker-compose -f local.yml up django
```

Now test on http://localhost:8000/2020/01/01/0/0/0/tile.png if you see the world mapnik like below.

Fig. 2.1: world map

To check if the demo data was set up right, go to http://localhost:8000/YEAR/MONTH/DAY/11/57/1134/tile.png. But change **YEAR**, **MONTH** & **DAY** to your current day. For example the like for the 2020-05-27 will be http://localhost:8000/2020/05/27/11/57/1134/tile.png The result should look like the tile below.

Fig. 2.2: niue

CHAPTER

THREE

GETTING UP AND RUNNING REMOTE WITH DOCKER

3.1 About

The project is Docker based. That means, the development requirements are just Docker and Docker Compose. Every other dependency will be automatically installed in a container. So the host system stays clean and every developer has the same dependencies!

Since the project need some beefy hardware and a strong network connection, it's recommended to use a *Remote Server*. But it's also possible to set up everything on the *Localhost*.

To start developing fast, it's recommended to use VS Code as IDE. Facebook & Microsoft created an extension to Remote Development, with this extension is it possible to use a remote server like it is in front of you.

3.2 Localhost

3.3 Remote Server

Note: This tutorial is tested on `Ubuntu 18.04`, it should also work under other Debian server.

3.3.1 Tested server hoster

Hoster	Status
Strato VServer	Unstable: threats limits
Digital Ocean	Works fine
Hetzner Cloud	Works fine
Netcup.de	Works fine

3.3.2 Setup server

At first connect to your remote server via SSH like.:

```
$ ssh root@your-server-ip
```

Update the server.:

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
```

Create a new user (in this case foo) and add him to the sudo group.:

```
$ adduser foo
$ adduser foo sudo
```

Log into your new user.:

```
$ su foo
$ cd
```

Add to ~/.ssh/authorized_keys your ssh public key. If you don't have an SSH key, go to Ubuntu Wiki OpenSSH to see how to create one.:

```
$ mkdir ~/.ssh
$ nano ~/.ssh/authorized_keys
```

Secure SSH to only allow to log in as non-root user and via ssh key.:

```
$ sudo nano /etc/ssh/sshd_config
```

Change PermitRootLogin yes to PermitRootLogin no and PasswordAuthentication yes to PasswordAuthentication no. Then press F2 and y and enter to save the file.

After changing the file, reload the SSH service.:

```
$ sudo /etc/init.d/ssh reload
```

Setup firewall with ufw, for a detail instruction go to Ubuntu Wiki firewall.:

```
$ sudo apt-get install ufw
$ sudo ufw default allow
$ sudo ufw allow ssh
$ sudo ufw allow 22
$ sudo ufw deny 4243
$ sudo ufw enable
$ sudo ufw status
```

Next install Docker, this is a quick instruction, for a more complex instruction go to docs.docker.com/install!:

```
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common \
    python-pip \
    python-setuptools \
```

(continues on next page)

(continued from previous page)

```
    python3-pip \
    python3-setuptools
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo apt-key fingerprint 0EBFCD88
$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Test if docker works.:

```
$ sudo docker run hello-world
```

Install docker compose.:

```
$ sudo pip3 install docker-compose
```

Add user `foo` to docker group, to run docker commands without `sudo`.:

```
$ sudo usermod -aG docker foo
```

Logout & login again to enable the changes. Then test if the user can use docker commands.:

```
$ docker run hello-world
```

Enable the docker API for localhost. For that edit the file `/lib/systemd/system/docker.service` and change the line beginning with `ExecStart=` to `ExecStart=/usr/bin/dockerd -H fd:// -H tcp://localhost:4243`.:

```
$ sudo nano /lib/systemd/system/docker.service
    #  change ExecStart= -> ExecStart=/usr/bin/dockerd -H fd:// -H tcp://
↪localhost:4243
$ sudo systemctl daemon-reload
$ sudo systemctl restart docker
```

To test if the api access works, create a http request.:

```
$ curl -X GET http://localhost:4243/images/json
[{"Containers":-1,"Created":1546306167,"Id":
↪"sha256:fce289e99eb9bca977dae136fbe2a82b6b7d4c372474c9235adc1741675f587e","Labels
↪":null,"ParentId":"","RepoDigests":["hello-
↪world@sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f"],
↪"RepoTags":["hello-world:latest"],"SharedSize":-1,"Size":1840,"VirtualSize":1840}]
```

Next setup `GIT`. To install just use `apt-get`.:

```
$ sudo apt-get install git
```

To configure git use.:

```
$ git config --global user.name "user_name"
$ git config --global user.email "your_email@example.com"
```

Create a new github SSH key, for deployment new commits.:

```
$ ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Add your new generated key to github.com.:

```
$ cat ~/.ssh/id_rsa.pub
```

Add the content of `~/.ssh/id_rsa.pub` to https://github.com/settings/keys.

At last, download the git repo via `SSH`. You can use the official repo `git@github.com:OpenHistoricalDataMap/MapnikTileServer.git` or use your own fork.:

```
$ git clone git@github.com:OpenHistoricalDataMap/MapnikTileServer.git ~/
↪MapnikTileServer
```

Also download the OHDM version of openstreetmap-carto.:

```
$ git clone git@github.com:linuxluigi/openstreetmap-carto.git ~/openstreetmap-carto
```

Now the server is ready to work :)

3.3.3 Setup VS Code

At first download & install VS Code for your desktop OS.

To work on a remote server, install the official Remote Development app. Next configure the access to the remote host, for that open in VS Code. For that click in the left bottom of VS Code on the remote extension.

If you need more information, go to the official docs.

Then select `Remote-SSH: Open Configuration File...`

Select your configuration file and then set up your host.:

```
Host HostShortName
    HostName HostIpAddress
    User foo
    LocalForward 127.0.0.1:4243 127.0.0.1:4243
    LocalForward 127.0.0.1:5432 127.0.0.1:5432
    LocalForward 127.0.0.1:5500 127.0.0.1:5500
    LocalForward 127.0.0.1:5555 127.0.0.1:5555
    LocalForward 127.0.0.1:8000 127.0.0.1:8000
```

After saving the file, you can now connect to your host via the remote extension.

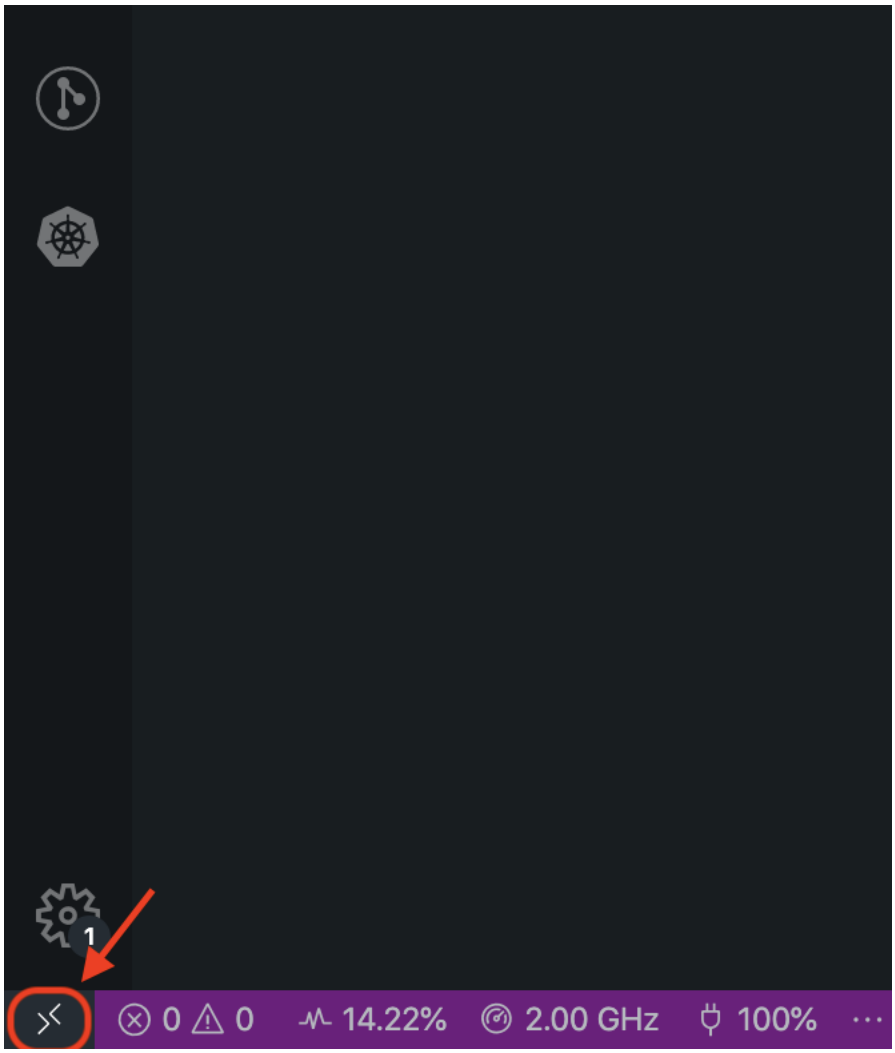

Fig. 3.1: VS Code - use remote extension

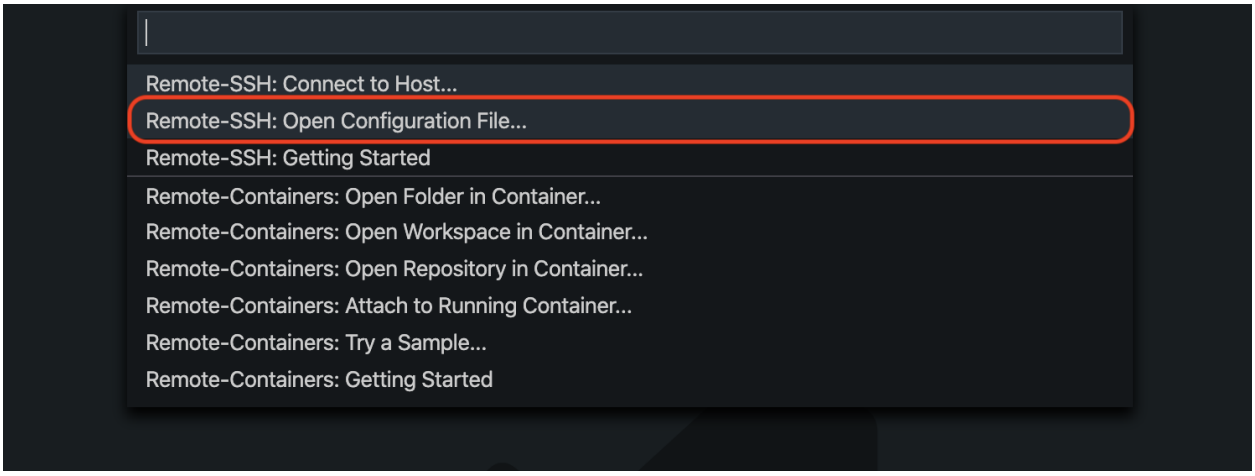

Fig. 3.2: VS Code - select `Open Configuration File`

CHAPTER

FOUR

COMMANDS

4.1 ohdm2mapnik

4.1.1 Intro

ohdm2mapnik convert a OHDM database schema to a mapnik readable Osm2pgsql schema.

4.1.2 Database setup

Note: This project does not contain an option to create a time sensitive OSM database. For this purpose use https://github.com/OpenHistoricalDataMap/OSMImportUpdate !

For this command a *OHDM* database connection is need to set up in `.envs/.local/.postgres` for the developing instance and for production is it `.envs/.production/.postgres`.

An example will look like:

```
# OHDM PostgreSQL
# ------------------------------------------------------------------------------
OHDM_SCHEMA=ohdm
```

4.1.3 Theory

For converting the database from OHDM schema to Osm2pgsql (mapnik readable) schema, the command will create an SQL statement, which will merge multiple OHDM tables into one output.

The tables are `classification`, `geoobject`, `geoobject_geometry` and one of `points`, `lines` or `polygon`, depending on the `type_target` in the `geoobject_geometry` table. So if the `type_target` is `0` or `1` it will create points, for 2 lines and 3 it will create polygons.

In Fig. 4.1 is an explanation, which data are use for merging the tables.

As next step, the `z_order` will be computed through the given data. The `z_order` is use in mapnik to order the objects for the renderer, so that an object with a higher `z_order` will be overdrawn an object with a lower `z_order`. To compute the `z_order`, every classification entry & every tag in `tags` will be gone a dict, where is defined how to rank an object. In Fig. 4.2 is an diagram how to calc the `z_order`.

In the same time, when compute the `z_order`, the system check if the dict which contains the `z_order` values, has a value for `is_road`. If this is `true`, a new `PlanetOsmRoads` object will be created from the data in the previous generated osm object. In Fig. 4.3 is a diagram how a `PlanetOsmRoads` object will be created.

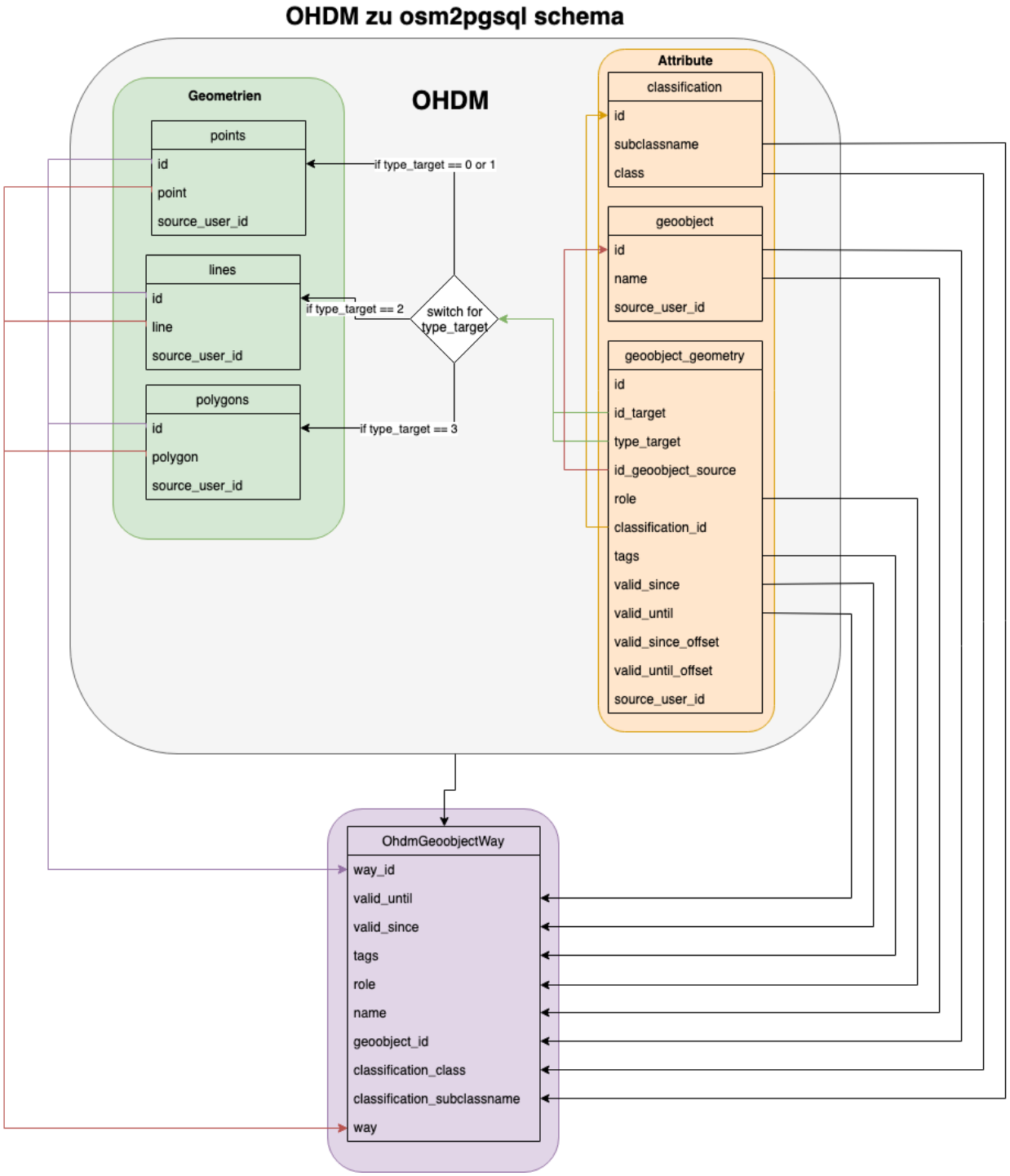

Fig. 4.1: Merge ohdm tables into one output

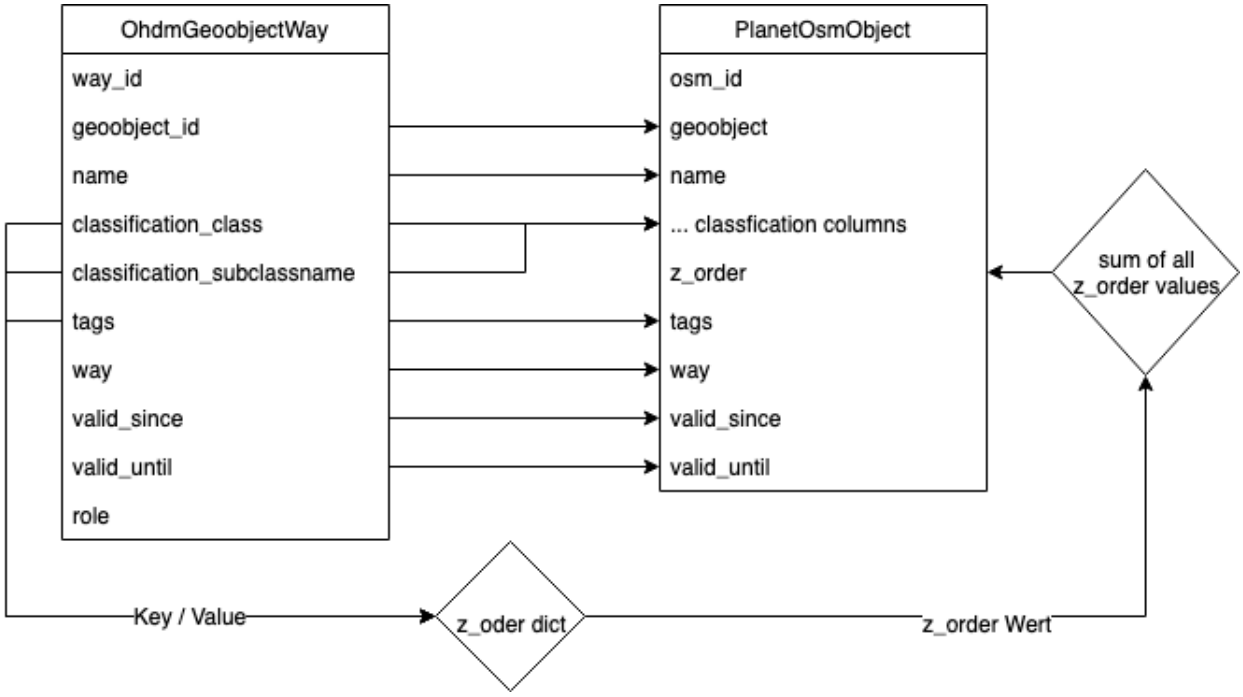

Fig. 4.2: Calc z_order

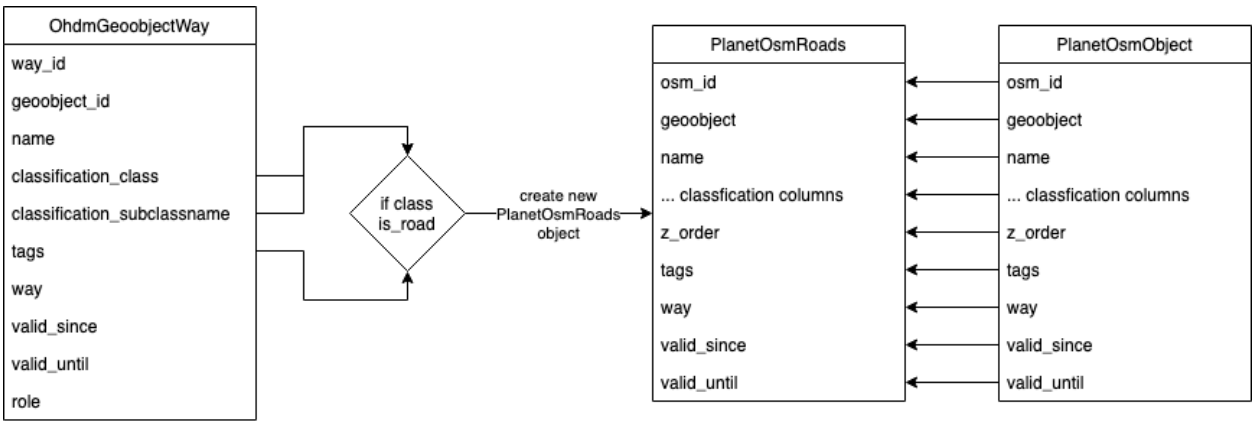

Fig. 4.3: planet_osm_roads

The code of the converter is mostly in ohdm2mapnik.py.

4.1.4 Usage

For production instance use

```
$ docker-compose -f production.yml run --rm django python manage.py ohdm2mapnik
```

For local instance use

```
$ docker-compose -f local.yml run --rm django python manage.py ohdm2mapnik
```

Optional parameters

--clear_mapnik_db Clear mapnik (osm2pgsql) data & tile cache

–cache [CACHE] Amount of object which will be handle at once!

--convert_points Points convert will be enabled, if set, only enabled geometries will be converted. By default, all geometries will be converted.

--convert_lines Lines convert will be enabled, if set, only enabled geometries will be converted. By default, all geometries will be converted.

--convert_polygons Polygons convert will be enabled, if set, only enabled geometries will be converted. By default, all geometries will be converted.

–sql_threads [SQL_THREADS] How many threats should be use, to insert entries into the database.

--not_fill_ohdm_tables Do not fill the ohdm cache table. Do this only if the ohdm cache tables already filled!

Hint: To reset just the mapnik tables (`planet_osm_*`) use `docker-compose -f local.yml run --rm django python manage.py migrate ohdm zero`. For faster database testing!

4.2 import_osh

Warning: This command is in experimental state, don't use this in production!

4.2.1 Usage

To import a OpenStreetMap history file (osh), use:

```
$ docker-compose -f local.yml run --rm django python manage.py import_osh --clear_rel_
↪db --clear_mapnik_db --rel2pgsql --cache 100000 --planet /osm-files/osh-file.osh.pbf
```

This command will import an osh file to the relation tables and after that, it will convert the relation tables to mapnik tables. It is possible to split this into two command.

To import an osh file into relation tables.:

```
$ docker-compose -f local.yml run --rm django python manage.py import_osh --planet /
↪osm-files/osh-file.osh.pbf
```

Note: Multipolygons are not integrated!

To convert relation tables to mapnik tables.:

```
$ docker-compose -f local.yml run --rm django python manage.py import_osh --rel2pgsql
```

4.2.2 Optional parameters

--clear_rel_db Clear relation data

--clear_mapnik_db Clear mapnik (osm2pgsql) data & tile cache

–cache [CACHE] Amount of object which will be handle at once!

--cache2file Cache osmium extraction into a file instead of memory

–planet [PLANET] Path to the planet file.

4.2.3 Get osh file

To download a history file for a region like germany, use https://www.geofabrik.de/

To import a complete planet go to https://wiki.openstreetmap.org/wiki/Planet.osm and choose a mirror, which support history files.

The osh file should be downloaded into `../osm-files/`, in the docker environment the folder `../osm-files/` will be mounted on `/osm-files/`. So to import the file `../osm-files/germany.osh` use:

```
$ docker-compose -f local.yml run --rm django python manage.py import_osh --planet /
↪osm-files/germany.osh
```

Because the importer is based on osmium, the file can be compressed with `bz2` and `pbf`.

4.3 import_osm

4.3.1 Usage

To import a OpenStreetMap file (osm), use:

```
$ docker-compose -f local.yml run --rm django python manage.py import_osm --clear_
↪mapnik_db --planet /osm-files/osm-file.osm.bz2 --cache 100000
```

This command will import an osm file into mapnik tables.

Optional parameters

--clear_mapnik_db Clear mapnik (osm2pgsql) data & tile cache

–cache [CACHE] Amount of object which will be handle at once!

--cache2file Cache osmium extraction into a file instead of memory

–planet [PLANET] Path to the planet file.

4.3.2 Get osm file

To download an osm file for a region like germany, use https://www.geofabrik.de/

To import a complete planet go to https://wiki.openstreetmap.org/wiki/Planet.osm and choose a mirror.

The osm file should be downloaded into `../osm-files/`, in the docker enviroment the folder `../osm-files/` will be mounted on `/osm-files/`. So to import the file `../osm-files/germany-latest.osm.bz2` use:

```
$ docker-compose -f local.yml run --rm django python manage.py import_osm --planet /
↪osm-files/germany-latest.osm.bz2
```

Because the importer is based on osmium, the file can be compressed with `bz2` and `pbf`.

4.4 Create Mapnik Style XML

To debug the `project.mml` from `openstreetmap-carto`, use:

```
$ docker-compose -f local.yml run --rm django python manage.py create_sytle_xml
```

This command will create a new mapnik `sytle.xml` with `carto`. Only useful when debugging `project.mml`.

4.5 set_indexes

To speedup the tile rendering, set indexes on the osm tables.:

```
$ docker-compose -f local.yml run --rm django python manage.py set_indexes
```

4.6 prerender

This will be prerendering all tile to the selected zoom level. To use the prerendering command use:

```
$ docker-compose -f local.yml run --rm django python manage.py prerender [ZOOM_LEVEL]
```

Change `[ZOOM_LEVEL]` to a value between `0` to `19`. A good value will be between `5` to `13`. The time to prerender all the tiles rise on each level times 4. This could take a large time to prerender all tiles!

4.7 clear_cache

Danger: Use this command only when you know what you do! This can't be undone!

With `docker-compose -f local.yml run --rm django python manage.py clear_cache` the complete cache will be deleted. Cached files are rendered Tile, previous called URL's and mapnik style XML for each date.

This is useful when you updated the Mapnik Tables and want to render the new data.

CHAPTER

FIVE

FRONTEND

The demo frontend is written as an Angular app. The source code is in an extra git repo https://github.com/linuxluigi/ohdm-angular-frontend

For handling the tile request to the MapntikTileServer it uses the library OpenLayers. The source code for the map handling is in map.component.ts.

Fig. 5.1: angular frontend

For faster tile request, use multiple domains. Each web browser has a limit how many `HTTP` connection can be use at once. To extend the limits, use multiple domains. In OpenLayers you can set a multiple domains in an array with `OlXYZ`.:

```
this.source = new OlXYZ({
    urls: [
        "https://a.ohdm.net/" + mapDate.year + "/" + mapDate.month + "/" + mapDate.
↪day + "/{z}/{x}/{y}/tile.png",
        "https://b.ohdm.net/" + mapDate.year + "/" + mapDate.month + "/" + mapDate.
↪day + "/{z}/{x}/{y}/tile.png",
        "https://c.ohdm.net/" + mapDate.year + "/" + mapDate.month + "/" + mapDate.
↪day + "/{z}/{x}/{y}/tile.png",
    ],
});
```

CHAPTER

SIX

SETTINGS

This project relies extensively on environment settings which **will not work with Apache/mod_wsgi setups**. It has been deployed successfully with both Gunicorn/Nginx and even uWSGI/Nginx.

For configuration purposes, the following table maps environment variables to their Django setting and project settings:

Environment Variable	Django Setting	Development Default	Production Default
DJANGO_READ_DOT_ENV_FILE	READ_DOT_ENV_FILE	False	False

Environment Variable	Django Setting	Development Default	Production Default
DATABASE_URL	DATABASES	auto w/ Docker; postgres://project_slug w/o	raises error
DJANGO_DEBUG	DEBUG	True	False
DJANGO_TEST_URLS	TEST_URLS	DEBUG	n/a
DJANGO_SETTINGS_MODULE	DJANGO_SETTINGS_MODULE	config.settings.local	raises error -> config.settings.production
DJANGO_SECRET_KEY	SECRET_KEY	auto-generated	raises error
DJANGO_SECURE_BROWSER_XSS_FILTER	SECURE_BROWSER_XSS_FILTER	n/a	True
DJANGO_SECURE_SSL_REDIRECT	SECURE_SSL_REDIRECT	n/a	True
DJANGO_SECURE_CONTENT_TYPE_NOSNIFF	SECURE_CONTENT_TYPE_NOSNIFF	n/a	True
DJANGO_SECURE_FRAME_DENY	SECURE_FRAME_DENY	n/a	True
DJANGO_SECURE_HSTS_INCLUDE_SUBDOMAINS	SECURE_HSTS_INCLUDE_SUBDOMAINS	n/a	True
DJANGO_SESSION_COOKIE_HTTPONLY	SESSION_COOKIE_HTTPONLY	n/a	True
DJANGO_SESSION_COOKIE_SECURE	SESSION_COOKIE_SECURE	n/a	False
DJANGO_DEFAULT_FROM_EMAIL	DEFAULT_FROM_EMAIL	n/a	"OHDM Django Mapnik <noreply@ohdm.net>"
DJANGO_SERVER_EMAIL	SERVER_EMAIL	n/a	"OHDM Django Mapnik <noreply@ohdm.net>"
DJANGO_EMAIL_SUBJECT_PREFIX	EMAIL_SUBJECT_PREFIX	n/a	"[OHDM Django Mapnik]"
DJANGO_ALLOWED_HOSTS	ALLOWED_HOSTS	['*']	[a.ohdm.net,b.ohdm.net,c.ohdm.net]
CELERY_BROKER_URL	CELERY_BROKER_URL	auto w/ Docker; raises error w/o	raises error
SENTRY_DSN	SENTRY_DSN	n/a	False
DJANGO_SENTRY_LOG_LEVEL	SENTRY_LOG_LEVEL	n/a	logging.INFO
CARTO_STYLE_PATH	CARTO_STYLE_PATH	raises error	raises error
CARTO_STYLE_PATH_DEBUG	CARTO_STYLE_PATH_DEBUG	raises error	n/a
TILE_GENERATOR_SOFT_TIMEOUT	TILE_GENERATOR_SOFT_TIMEOUT	240	240
TILE_GENERATOR_HARD_TIMEOUT	TILE_GENERATOR_HARD_TIMEOUT	360	360
ZOOM_LEVEL	ZOOM_LEVEL	13	13
TILE_CACHE_TIME	TILE_CACHE_TIME	2592000	2592000

CHAPTER

SEVEN

CONFIG

There are 3 exiting configs for django.

Local The local config is for developing. In the config there is the debug mode by default enabled. The file is `config/settings/local.py`

Production The production config is for production mode. The file is `config/settings/production.py`

Testing The testing config is for pytest. The file is `config/settings/test.py`

CHAPTER

EIGHT

VIEW

To change how to handle a tile request, you need to modify the `ohdm_django_mapnik/ohdm/views.py`.

The default value to render a tile is the function `generate_tile`. Any other functions are only for debugging & developing.

If need to change the URL structure, modify the file `ohdm_django_mapnik/ohdm/urls.py`.

Note: Don't forget to test your changes with `docker-compose -f local.yml run --rm django pytest -s`

CHAPTER

NINE

CACHING

9.1 Basic

There are 3 tings, that will be cached.

1. Mapnik Style XML for each date
2. tile cache location
3. tile

The cache will be filled on each `HTTP` request, where there is no cached file or when using the prerendering command.:

```
$ docker-compose -f local.yml run --rm django python manage.py prerender [ZOOM_LEVEL]
```

Below is a diagram, how the cache is used in a view.

9.2 Caching Objects

9.2.1 1. Mapnik Style XML for each date

Cache will be created, when a tile process is running. For each date, the system need a different mapnik style XML.

9.2.2 2. tile cache location

Cache location of a tile. To save space on the cache server, every tile will be hashed with `MD5` and saved under the `MD5` value. When multiple tiles have the same `MD5` hash, then only one tile will be saved. The cache location key is `year-month-day-zoom-X-Y`.

9.2.3 3. tile

Finally, the tile PNG. The cache key is the hash value of the tile.

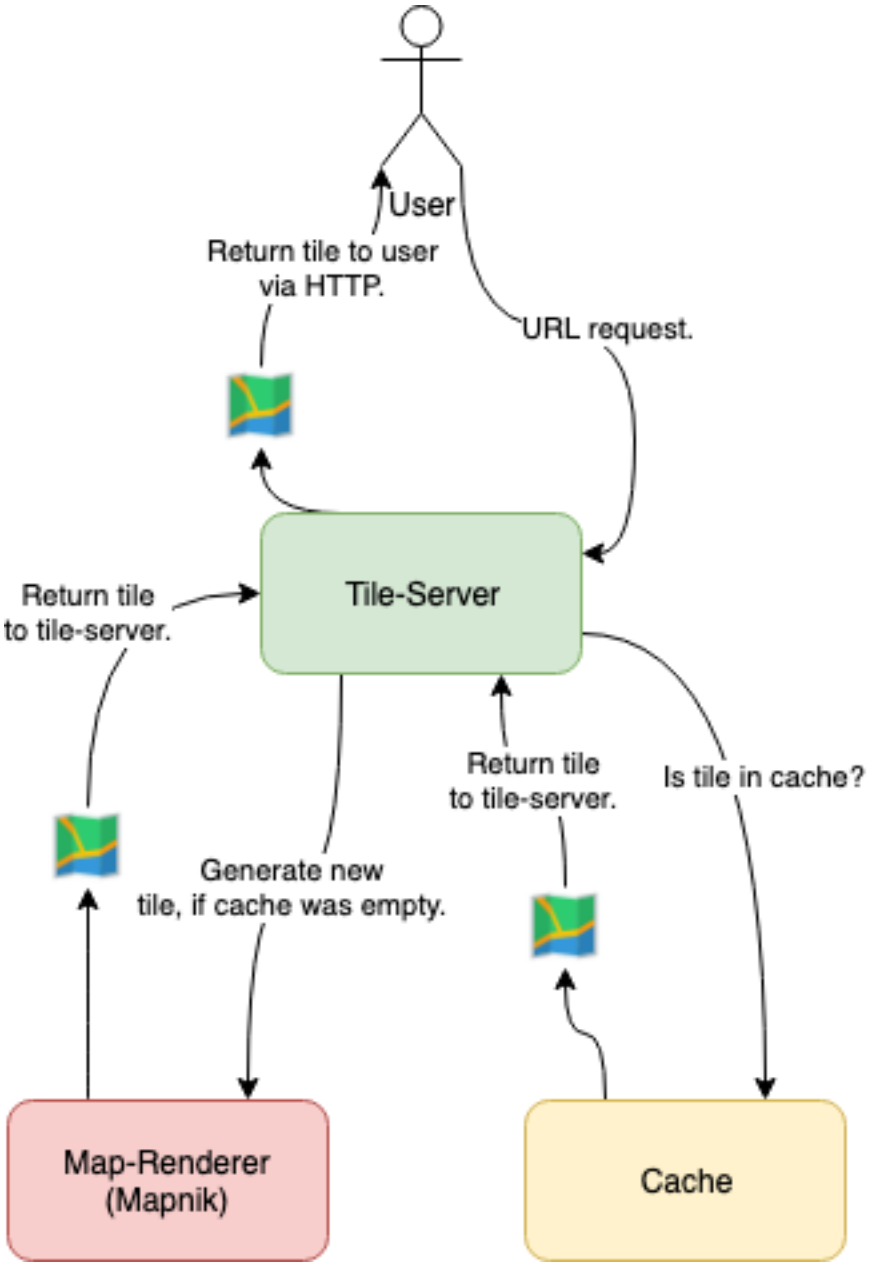

Fig. 9.1: Tile Server overview

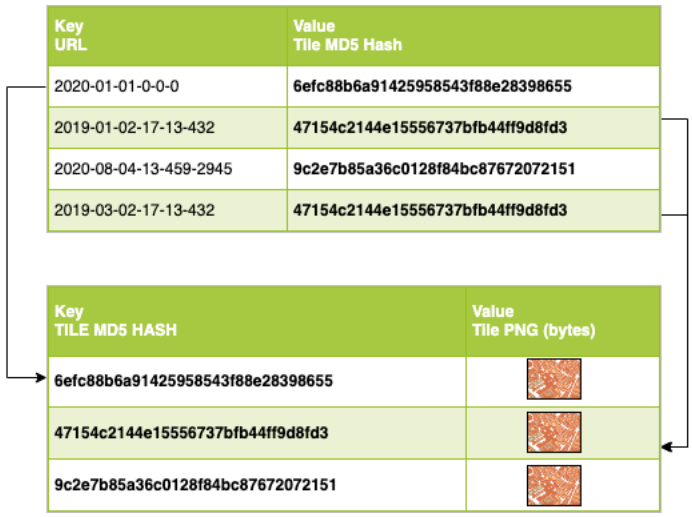

Fig. 9.2: tiles of a map

9.3 Config

In the config file, you can set, how low the cache should be saved.:

```
# caching
# ------------------------------------------------------------------------------
# to which zoom level tiles should cached for ever
ZOOM_LEVEL=13
# 2592000 == 1 month
TILE_CACHE_TIME=2592000
```

ZOOM_LEVEL to which zoom level the cache should be stored for ever.

TILE_CACHE_TIME how long a tile should be cached in seconds.

9.4 Caching Objects

To clear up all cached files use:

```
$ docker-compose -f local.yml run --rm django python manage.py clear_cache
```

But be careful, this command can't be undone!

9.5 Compression

To save space on the cache system, any cache object will be compressed with `lzma`. For changing the compression mode, modify `config/settings/production.py` under `CACHES`. On https://docs.python.org/3/library/lzma.html#module-lzma is a list, with all available compression modes.

CHAPTER

TEN

MONITORING

10.1 Flower

Flower is celery task monitor. Here you can watch, how the tiles are produced and if their any errors.

To access flower, you need to add the flower username & password in your `.envs/.production/.django` for poduction and `.envs/.local/.django` for the development instance.

```
# Celery / Flower # ————————————————————————————————————
CELERY_FLOWER_USER=msshnhBNlGfVLiDTErFKxFWpBOrcZNVp CEL-
ERY_FLOWER_PASSWORD=eIYplziix19gsYgGfu7HsVzlhHUwFJxQikoOwDlSDnYCjQEjo1atzobXZCyrmnG0
```

On the development instance, you can access flower over http://localhost:5555 In production, the default URL is https://monitor.ohdm.net If you want change it to a different domain, modify `compose/production/traefik/traefik.yml`

Read more at https://flower.readthedocs.io/en/latest/

10.2 Sentry.io

Sentry.io is an online monitoring platform for monitoring error. To enable it, add the enviroment var `SENTRY_DSN` in `.envs/.production/.django` with your account sentry DNS.

CHAPTER

ELEVEN

URL'S

The URL structure of the project is set up in `config/urls.py` and `ohdm_django_mapnik/ohdm/urls.py`.

In the file `config/urls.py` there are server wide URL's like `/admin` and in `ohdm_django_mapnik/ohdm/urls.py` there are just URL's for the tile server.

Admin panel URL: `/admin`

Time sensitive tile URL: `/tile/<int:year>/<int:month>/<int:day>/<int:zoom>/<float:x_pixel>/<float:y_pixel>/tile.png`

Only in development mode enabled!

Tile URL with reload style.xml `/tile/<int:year>/<int:month>/<int:day>/<int:zoom>/<float:x_pixel>/<float:y_pixel>/reload-style-xml/tile.png`

Tile URL with reload project.mml & style.xml `/tile/<int:year>/<int:month>/<int:day>/<int:zoom>/<float:x_pixel>/<float:y_pixel>/reload-project-mml/tile.png`

Tile URL with default openstreetmap-carto (no time sensitivity) `/tile/<int:zoom>/<float:x_pixel>/<float:y_pixel>/tile.png`

CHAPTER

TWELVE

OPENSTREETMAP-CARTO

The openstreetmap-carto project is the official repo for the style sheets of OpenStreetMap.org.

For this project there is a fork of openstreetmap-carto, which can handle the time request on Postgres SQL. The fork is hosted on github.com.

The only difference between the fork version and the original the file `/project.mml`. For each request is a where clause added, which check in the columns `valid_since` & `valid_until` if the request is for the current date. The data parameter is added by `{{ date.strftime('%Y-%m-%d') }}` and will be converted in a python script with jinja2 into the requested date. An example for a modified clause.:

```
(SELECT
    valid_since,
    valid_until,
    way
    FROM planet_osm_line
    WHERE valid_since <= '{{ date.strftime('%Y-%m-%d') }}'
        AND valid_until >= '{{ date.strftime('%Y-%m-%d') }}'
        AND (man_made = 'cutline')
) AS landcover_line
```

CHAPTER

THIRTEEN

SOURCE CODE

Since this project is split over multiple GitHub repos, here is a list of the important links:

- OHDM Github Project page: https://github.com/OpenHistoricalDataMap
- MapnikTileServer: https://github.com/OpenHistoricalDataMap/MapnikTileServer
- openstreetmap-carto: https://github.com/linuxluigi/openstreetmap-carto/
- Frontend: https://github.com/linuxluigi/ohdm-angular-frontend
- OSMImportUpdate (for creating a time sensitive OHDM database): https://github.com/OpenHistoricalDataMap/OSMImportUpdate

CHAPTER

FOURTEEN

TESTS

14.1 pytest

Tests are written with pytest.

The pytests stored in `ohdm_django_mapnik/ohdm/tests/`.

Before running the test, you need to fill the database with some test data, create a mapnik style XML and load shape files.:

```
$ docker-compose -f local.yml run --rm django /get-shapefiles.sh
$ docker-compose -f local.yml run --rm django python manage.py create_style_xml
$ docker-compose -f local.yml run --rm django python manage.py migrate
$ docker-compose -f local.yml run --rm django python manage.py import_osm --planet /
↪niue-latest.osm.pbf
```

To run the tests use.:

```
$ docker-compose -f local.yml run --rm django pytest
```

If you want to see the terminal output, you can add `-s`:

```
$ docker-compose -f local.yml run --rm django pytest -s
```

For testing a folder or single file, add the relative path to the folder or file.:

```
$ docker-compose -f local.yml run --rm django pytest ohdm_django_mapnik/ohdm/tests
$ docker-compose -f local.yml run --rm django pytest ohdm_django_mapnik/ohdm/tests/
↪test_tile.py
```

For testing a single test in a test file, add the function after `::`:

```
$ docker-compose -f local.yml run --rm django pytest ohdm_django_mapnik/ohdm/tests/
↪test_tile.py::test_tile_generator_init
```

14.2 Mypy

From mypy-lang.org:

> Mypy is an optional static type checker for Python that aims to combine the benefits of dynamic (or "duck") typing and static typing. Mypy combines the expressive power and convenience of Python with a powerful type system and compile-time type checking. Mypy type checks standard Python programs; run them using any Python VM with basically no runtime overhead.

On travis pipeline, the mypy tests will run and if their some error, the whole pipeline will show as `build failed`.

To run mypy:

```
$ docker-compose -f local.yml run --rm django mypy ./
```

CHAPTER

FIFTEEN

DEPLOYMENT

Note: This tutorial is only tested on Debian Buster!

15.1 Firewall

Ports `80` & `443` need to be open, for installing the dependencies & running the server. An example for iptables to open the ports.:

```
$ iptables -A INPUT -p tcp --dport 443 -j ACCEPT
$ iptables -A OUTPUT -p tcp --dport 443 -j ACCEPT
$ iptables -A INPUT -p tcp --dport 80 -j ACCEPT
$ iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT
```

15.2 Dependent Services

15.2.1 Postgres 12 with Postgis 3

To use the performance boost of the new Postgres Version, add the postgres repo from postgresql.org to your system.:

```
$ apt-get --no-install-recommends install \
    locales gnupg2 wget ca-certificates rpl pwgen software-properties-common gdal-bin␣
↪iputils-ping
$ sh -c "echo \"deb http://apt.postgresql.org/pub/repos/apt/ buster-pgdg main\" > /
↪etc/apt/sources.list.d/pgdg.list"
$ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key␣
↪add -
$ apt-get update
```

Install Postgres 12 with Postgis 3.:

```
$ apt-get --no-install-recommends install postgresql-client-12 \
    postgresql-common postgresql-12 postgresql-12-postgis-3 \
    netcat postgresql-12-ogr-fdw postgresql-12-postgis-3-scripts \
    postgresql-12-cron postgresql-plpython3-12 postgresql-12-pgrouting
```

Postgres's service is started & set to come up after each system reboot.:

```
$ systemctl status postgresql.service
```

While the installation, the user `postgres` was added to the system. With the user you can access the admin postgres user.

$ su - postgres

Change the postgres user `postgres` password (remember this password!):

```
$ psql -c "alter user postgres with password 'YourNewPassword'"
```

Now access the postgres prompt.:

```
$ psql
```

Enable Postgis & hstore extensions for postgres.:

```
$ CREATE EXTENSION postgis;
$ CREATE EXTENSION hstore;
$ CREATE EXTENSION postgis_topology;
```

Set the template postgres database to `UTF-8` encoding.:

```
$ UPDATE pg_database SET datistemplate = FALSE WHERE datname = 'template1';
$ DROP DATABASE template1;
$ CREATE DATABASE template1 WITH TEMPLATE = template0 ENCODING = 'UTF8';
$ UPDATE pg_database SET datistemplate = TRUE WHERE datname = 'template1';
$ \c template1
$ VACUUM FREEZE;
```

Create the `gis` database with the user `mapnik` to access the `gis` database.:

```
$ CREATE DATABASE gis;
$ CREATE USER mapnik WITH ENCRYPTED PASSWORD 'MyStr0ngP@SS';
$ GRANT ALL PRIVILEGES ON DATABASE gis to mapnik;
```

Set the new `mapnik` database user as superuser.:

```
$ ALTER USER mapnik WITH SUPERUSER;
```

Logout from postgres prompt & user.:

```
$ \q
$ exit
```

15.2.2 Redis 5

Redis use as a caching server for the tiles & as a task broker for celery.

For installing redis server, use:

```
$ apt-get install --no-install-recommends redis-server
```

If you running redis on the same system as the web-service, then is redis ready to work :)

15.2.3 NGINX

Install NGINX and certbot for Let's Encrypt.:

```
$ apt-get install --no-install-recommends nginx python3-acme \
    python3-certbot python3-mock python3-openssl python3-pkg-resources \
    python3-pyparsing python3-zope.interface python3-certbot-nginx
```

Create a NGINX config file for ohdm.:

```
$ nano /etc/nginx/sites-available/MapnikTileServer.conf

server {
    server_name a.ohdm.net b.ohdm.net c.ohdm.net;

    location = /favicon.ico { access_log off; log_not_found off; }
    location  /static/ {
    alias /home/mapnik/MapnikTileServer/staticfiles/;
    }

    location / {
        include proxy_params;
        proxy_read_timeout 360;
        proxy_pass http://unix:/home/mapnik/MapnikTileServer/MapnikTileServer.sock;
    }

}

server {
    server_name monitor.ohdm.net;

    location / {
        include proxy_params;
        proxy_pass http://127.0.0.1:5555;
    }
}
```

Note: Change the domains `a.ohdm.net`, `b.ohdm.net`, `c.ohdm.net` & `monitor.ohdm.net` in the NGINX config file to your domains!

Link the config file from `/etc/nginx/sites-available/MapnikTileServer.conf` to `/etc/nginx/sites-enabled/MapnikTileServer.conf`.:

```
$ ln -s /etc/nginx/sites-available/MapnikTileServer.conf /etc/nginx/sites-enabled
```

Test if the config was set up right & restart NGINX.:

```
$ nginx -t
$ systemctl restart nginx
```

Obtaining an SSL Certificate.:

```
$ certbot --nginx -d a.ohdm.net -d b.ohdm.net -d c.ohdm.net -d monitor.ohdm.net
2
2
```

Test if Let's Encrypt was sucessfully set up.:

```
$ nginx -t
$ systemctl restart nginx
```

Test if certbot can auto renew the SSL certificate.:

```
$ certbot renew --dry-run
```

15.3 Install MapnikTileServer

System dependencies:

```
$ apt-get install --no-install-recommends wget unzip fontconfig gnupg
```

Node:

```
$ apt-get install nodejs npm
$ npm i -g npm@^6
```

Python:

```
$ apt-get install --no-install-recommends python3-pip python3-dev \
    python3-setuptools
```

Mapnik-utils for openstreetmap-carto:

```
$ apt-get install --no-install-recommends mapnik-utils
```

Dependencies for building Python packages:

```
$ apt-get install --no-install-recommends build-essential
```

Psycopg2 dependencies:

```
$ apt-get install --no-install-recommends libpq-dev
```

Translations dependencies:

```
$ apt-get install --no-install-recommends gettext
```

Fonts for mapnik:

```
$ apt-get install --no-install-recommends fonts-dejavu fonts-hanazono \
ttf-unifont \
fonts-noto fonts-noto-cjk fonts-noto-cjk-extra fonts-noto-color-emoji \
fonts-noto-hinted fonts-noto-mono \
fonts-noto-unhinted \
fonts-noto-extra fonts-noto-ui-core fonts-noto-ui-extra
```

Geodjango:

```
$ apt-get install --no-install-recommends binutils libproj-dev gdal-bin
```

Git:

```
$ apt-get install --no-install-recommends git
```

Mapnik:

```
$ apt-get install --no-install-recommends libmapnik-dev libmapnik3.0 mapnik-utils \
python3-mapnik
```

Supervisor:

```
$ apt-get install --no-install-recommends supervisor
```

Download & install more noto fonts for mapnik:

```
$ mkdir noto-fonts
$ cd noto-fonts
$ wget https://noto-website-2.storage.googleapis.com/pkgs/NotoSansBalinese-unhinted.
↪zip
$ wget https://noto-website-2.storage.googleapis.com/pkgs/NotoSansSyriacEastern-
↪unhinted.zip
$ wget https://noto-website-2.storage.googleapis.com/pkgs/NotoColorEmoji-unhinted.zip
$ wget https://noto-website-2.storage.googleapis.com/pkgs/NotoEmoji-unhinted.zip
$ unzip -o \*.zip
$ cp ./*.ttf /usr/share/fonts/truetype/noto/
$ fc-cache -fv
$ fc-list
$ cd ..
$ rm -r noto-fonts
```

Update NodeJS to the latest stable:

```
$ npm install -g n stable
```

Install CartoCSS with a version below 1.:

```
$ npm install -g carto@0
```

Set environment vars for running the MapnikTileServer.:

```
$ nano /etc/environment
```

Fill the `/etc/environment` file with the following values.

```
# Django DJANGO_READ_DOT_ENV_FILE=True DJANGO_SETTINGS_MODULE=config.settings.production
```

Create a Mapnik user, for running the MapnikTileServer.:

```
$ adduser mapnik
```

Log into `mapnik` user and go to the home folder.:

```
$ su - mapnik
$ cd
```

Download openstreetmap-carto:

```
$ git clone https://github.com/linuxluigi/openstreetmap-carto.git
```

Go to the new openstreetmap-carto folder, download the shape files & create the default mapnik style XML:

```
$ cd openstreetmap-carto
$ ./scripts/get-shapefiles.py
$ carto project.mml > style.xml
```

Next go back to the `mapnik` home folder.:

```
$ cd
```

Download MapnikTileServer.:

```
$ git clone https://github.com/OpenHistoricalDataMap/MapnikTileServer.git
$ cd MapnikTileServer
```

Install / update the python packages as root user.:

```
$ exit
$ pip3 install -r /home/mapnik/MapnikTileServer/requirements/system.txt
$ pip3 install -r /home/mapnik/MapnikTileServer/requirements/base.txt
$ pip3 install -r /home/mapnik/MapnikTileServer/requirements/production.txt
```

Note: When install an update of MapnikTileServer, also update the python packages!

Go back to the `mapnik` user & back to the MapnikTileServer folder.:

```
$ su mapnik
$ cd /home/mapnik/MapnikTileServer
```

Create a `.env` file for the MapnikTileServer settings. Go to *Settings* to see all possibles options. Below is a minimal configuration:

```
# General
# ------------------------------------------------------------------------------
DJANGO_SECRET_KEY=!!!ChangeMeToSomeRandomValue!!!!!
DJANGO_ALLOWED_HOSTS=a.ohdm.net,b.ohdm.net,c.ohdm.net

# Redis
# ------------------------------------------------------------------------------
REDIS_URL=redis://localhost:6379/0
CELERY_BROKER_URL=redis://localhost:6379/0

# ohdm
# ------------------------------------------------------------------------------
CARTO_STYLE_PATH=/home/mapnik/openstreetmap-carto

# Default PostgreSQL
# ------------------------------------------------------------------------------
DATABASE_URL="postgres://mapnik:MyStr0ngP@SS@localhost:5432/gis"
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=gis
POSTGRES_USER=mapnik
POSTGRES_PASSWORD=MyStr0ngP
PGCONNECT_TIMEOUT=60

# OHDM PostgreSQL
```

(continues on next page)

(continued from previous page)

```
# ------------------------------------------------------------------------------
OHDM_SCHEMA=ohdm
```

Tests the settings, migrate the database, set indexes & collect static files:

```
$ python3 manage.py migrate
$ python3 manage.py set_indexes
$ python3 manage.py collectstatic
```

Add a superuser for the admin panel.:

```
$ python3 manage.py createsuperuser
```

Add `supervisor` script to auto start django, celery & flower at system start. For creating the scripts, go back to the root user.:

```
$ exit
```

Open the text editor to create the `supervisor` file.:

```
$ nano /etc/supervisor/conf.d/mapnik_tile_server.conf
```

Fill the `supervisor` file with the values below, but don't forget to change `--basic_auth="ChangeMeFlowerUser:ChangeMeFlowerPassword"` with your flower user & password.:

```
[supervisord]
environment=DJANGO_READ_DOT_ENV_FILE=True,DJANGO_SETTINGS_MODULE=config.settings.
↪production,CELERY_BROKER_URL=redis://localhost:6379/0

[program:MapnikTileServer_celery_worker]
command=celery -A config.celery_app worker -l INFO
user=mapnik
directory=/home/mapnik/MapnikTileServer
autostart=true
autorestart=true
priority=10
stderr_logfile=/var/log/MapnikTileServer_celery_worker.err.log

[program:MapnikTileServer_celery_beat]
command=celery -A config.celery_app beat -l INFO
user=mapnik
directory=/home/mapnik/MapnikTileServer
autostart=true
autorestart=true
priority=10
stderr_logfile=/var/log/MapnikTileServer_celery_beat.err.log

[program:MapnikTileServer_celery_flower]
command=celery flower --app=config.celery_app --broker="redis://localhost:6379/0" --
↪basic_auth="ChangeMeFlowerUser:ChangeMeFlowerPassword"
user=mapnik
directory=/home/mapnik/MapnikTileServer
autostart=true
autorestart=true
priority=10
```

(continues on next page)

(continued from previous page)

```
stderr_logfile=/var/log/MapnikTileServer_celery_flower.err.log

[program:MapnikTileServer_django]
command=/usr/local/bin/gunicorn config.wsgi --workers 2 --bind unix:/home/mapnik/
↪MapnikTileServer/MapnikTileServer.sock -t 360
user=mapnik
directory=/home/mapnik/MapnikTileServer
autostart=true
autorestart=true
priority=10
stderr_logfile=/var/log/MapnikTileServer_django.err.log
```

To enable the `supervisor` script.:

```
$ supervisorctl reread
$ supervisorctl update
$ supervisorctl start all
$ supervisorctl status
```

15.4 Use commands

For using django commands from *Commands*, log into the `mapnik` user & go to the `/home/mapnik/MapnikTileServer`.:

```
$ su mapnik
$ cd /home/mapnik/MapnikTileServer
```

The commands in *Commands* are written for the docker usage, to use them without docker, just use the command after the `django` keyword. For example, to use `set_indexes`, in the docs the command is written down as `docker-compose -f local.yml run --rm django python manage.py set_indexes` and to use it without docker, just use `python3 manage.py set_indexes`.

15.5 Download updates

Stop all services first.:

```
$ supervisorctl stop all
```

Log into the `mapnik` user and go to the openstreetmap-carto folder.:

```
$ su mapnik
$ cd /home/mapnik/openstreetmap-carto
```

Get the latest version with `git pull`.:

```
$ git pull
```

Download the latest shape files & create the default mapnik style XML.:

```
$ ./scripts/get-shapefiles.py
$ carto project.mml > style.xml
```

Go to the MapnikTileServer.:

```
$ cd /home/mapnik/MapnikTileServer
```

Download the latest code from github, for the MapnikTileServer.:

```
$ git pull
```

Update the database & static files.:

```
$ python3 manage.py migrate
$ python3 manage.py set_indexes
$ python3 manage.py collectstatic
```

Log out from the `mapnik` user & start the web services again.:

```
$ exit
$ supervisorctl start all
```

Remove all packages were automatically installed and are no longer required.:

```
$ apt autoremove
```

On `monitor.ohdm.net` you should now able to log into the flower celery monitor. The user information is in `/etc/supervisor/conf.d/mapnik_tile_server.conf` on the line `--basic_auth="ChangeMeFlowerUser:ChangeMeFlowerPassword"`.

CHAPTER

SIXTEEN

DEPLOYMENT WITH DOCKER

Note: The article is based on https://cookiecutter-django.readthedocs.io/en/latest/deployment-with-docker.html

16.1 Prerequisites

- Docker 17.05+.
- Docker Compose 1.17+

16.2 Download the software

To download the source, use git with.:

```
$ git clone https://github.com/OpenHistoricalDataMap/MapnikTileServer.git
```

16.3 Configuration with environment vars

To set up the server for your personal needs, you need to create 2 files. `.envs/.production/.django` & `.envs/.production/.postgres`.

All possible environment vars are list in *Settings*. A minimal config file `.envs/.production/.django` will look like:

```
# General
# ------------------------------------------------------------------------------
USE_DOCKER=yes
IPYTHONDIR=/app/.ipython
DJANGO_SECRET_KEY=!!!ChangeMeToSomeRandomValue!!!!!
DJANGO_ALLOWED_HOSTS=a.ohdm.net,b.ohdm.net,c.ohdm.net
DJANGO_SETTINGS_MODULE=config.settings.production

# Redis
# ------------------------------------------------------------------------------
REDIS_URL=redis://redis:6379/0

# ohdm
```

(continues on next page)

(continued from previous page)

```
# ------------------------------------------------------------------------------
CARTO_STYLE_PATH=/opt/openstreetmap-carto
```

In the `.envs/.production/.postgres` file is the connection to the PostGis server and the schema of the OHDM data included. For the minimal configuration.:

```
# Default PostgreSQL
# ------------------------------------------------------------------------------
POSTGRES_HOST=postgres
POSTGRES_PORT=5432
POSTGRES_DB=gis
POSTGRES_USER=docker
POSTGRES_PASSWORD=Dyx8lXMKIGggiQXTzSrAuZ3UsDt8YmLy53WEIAga6EkkVc2GK9lmiRfJxzx7Oahw
POSTGRES_MULTIPLE_EXTENSIONS=postgis,hstore,postgis_topology
PGCONNECT_TIMEOUT=60

# OHDM PostgreSQL
# ------------------------------------------------------------------------------
OHDM_SCHEMA=ohdm
```

More possible options can be read on docker-postgis.

16.3.1 Need to change

DJANGO_SECRET_KEY Need to be randomly unique value to provide cryptographic signing. For creating a randomly-generated SECRET_KEY, you can use https://djecrety.ir/

Docs: https://docs.djangoproject.com/en/3.0/ref/settings/#std:setting-SECRET_KEY

DJANGO_ALLOWED_HOSTS Domains for the django container, add all domains which are pointed to the container, separated by `,`.

16.3.2 Monitoring

There a two monitoring system integrated. Flower monitor the tile producing queue and Sentry.io for logging all errors on sentry.io. For setup go to *Monitoring*.

16.4 Changing the default domains

To change the default domains, you need to modify `compose/production/traefik/traefik.yml`` and set the enviroment var `DJANGO_ALLOWED_HOSTS`.

16.5 Building & Running Production Stack

You will need to build the stack first. To do that, run:

```
$ docker-compose -f production.yml build
```

Once this is ready, you can run it with:

```
$ docker-compose -f production.yml up
```

To run the stack and detach the containers, run:

```
$ docker-compose -f production.yml up -d
```

To run a migration, open up a second terminal and run:

```
$ docker-compose -f production.yml run --rm django python manage.py migrate
```

To create a superuser, run:

```
$ docker-compose -f production.yml run --rm django python manage.py createsuperuser
```

If you need a shell, run:

```
$ docker-compose -f production.yml run --rm django python manage.py shell
```

To check the logs out, run:

```
$ docker-compose -f production.yml logs
```

If you want to scale your application, run:

```
$ docker-compose -f production.yml scale django=70
$ docker-compose -f production.yml scale celeryworker=2
```

> **Warning:** don't try to scale `postgres`, `celerybeat`, or `traefik`.

To see how your containers are doing run:

```
$ docker-compose -f production.yml ps
```

16.6 Example: Supervisor

Once you are ready with your initial setup, you want to make sure that your application is run by a process manager to survive reboots and auto restarts in case of an error. You can use the process manager you are most familiar with. All it needs to do is to run `docker-compose -f production.yml up` in your projects root directory.

If you are using `supervisor`, you can use this file as a starting point:

```
[program:MapnikTileServer]
command=docker-compose -f production.yml up
directory=/path/to/MapnikTileServer
```

(continues on next page)

(continued from previous page)

```
redirect_stderr=true
autostart=true
autorestart=true
priority=10
```

Move it to `/etc/supervisor/conf.d/MapnikTileServer.conf` and run:

```
$ supervisorctl reread
$ supervisorctl update
$ supervisorctl start MapnikTileServer
```

For status check, run:

```
$ supervisorctl status
```

CHAPTER

SEVENTEEN

DOCUMENTATION

The documentation files are in `/docs` and written with Spinx-Doc in reStructuredText language. ReStructuredText is like Markdown with some extra features.

The docs can be build as HTML, PDF or E-PUB.

To build HTML files use:

```
$ docker-compose -f local.yml run --rm django make --directory docs html
```

The compiled HTML files are in `/docs/_build/html`.

When uploading the docs on https://readthedocs.io as a github hook, it will auto compile each version as HTML, PDF & E-PUB.

The latest version of the docs are on https://mapniktileserver.readthedocs.io/en/latest/?badge=latest

CHAPTER

EIGHTEEN

C/I

18.1 About

MapntikTileServer use Github.com Marketplace Apps to maintain the project. Every App is for free for Open Source projects!

18.2 Code Style

18.2.1 Black

Black is not integrated as a C/I, it's just a python code auto formatter for the project. So if you like to contribute your code use black by `python black ./`!

To format any file in this file in black, use:

```
$ docker-compose -f local.yml run --rm django black ./
```

18.3 Tests

18.3.1 Travis

This project use for testing unit test, django commands & Docker-Compose builds Travis

Travis config is `.travis.yml`.

18.4 Documentation

18.4.1 Readthedocs.org

Documentation is written in Sphinx in `.rst` file format. The sourcecode of the docs is in `docs/`.

Travis config is `.readthedocs.yml`.

18.5 Code Review

18.5.1 Codacy.com

Codacy.com is an automated code analysis/quality tool. Codacy analyze only python for this project, also the coverage of the test are uploaded to Codacy.com via Travis.

18.5.2 DeepSource.io

DeepSource.io is like Codacy.com but it also analyzes Dockerfiles.

DeepSource config is `.deepsource.toml`

18.6 Dependencies

18.6.1 Pyup.io

Pyup.io update Python packages once a week. It pushes every update to an extra branch & create a pull request.

Pyup config is `.pyup.yml`

18.6.2 Dependabot.com

Dependabot.com update Dockerfiles once a week. It pushes every update to an extra branch & create a pull request.

Dependabot config is `.dependabot/config.yml`

CHAPTER

NINETEEN

TROUBLESHOOTING

This page contains some advice about errors and problems commonly encountered during the development of Mapnik Tile Server.

19.1 Can't install Docker on Windows

To use Docker on Windows you need a PRO version.

19.2 Can't start docker container on Windows

When downloading the source code via GitHub Desktop it can happen, that every file is refactored for windows usage, but when try to run the code on a docker container (linux) it will crash!

So to solve the issue, try to download via the `CLI` or via VS Code.

19.3 bash: fork: retry

When developing on a *Remote Server*, it can happen that you get the error:

```
bash: fork: retry: Die Ressource ist zur Zeit nicht verfügbar
bash: fork: retry: Die Ressource ist zur Zeit nicht verfügbar
bash: fork: retry: Die Ressource ist zur Zeit nicht verfügbar
bash: fork: retry: Die Ressource ist zur Zeit nicht verfügbar
```

To solve this error, expand the process limits of your target user. For all users the command is:

```
$ echo '* soft nofile 65000' | sudo tee --append /etc/security/limits.conf
$ echo '* hard nofile 65000' | sudo tee --append /etc/security/limits.conf
$ sudo reboot
```

After the reboot, it shouldn't shown the error message again. If this message isn't gone after restart, you may need to use an another hoster. On *Tested server hoster* you can watch out for a new working hoster.

Please make sure, that the hoster is not overwrite the file on each restart!

19.4 unable to find face-name 'unifont Medium' in FontSet 'fontset-0'

If the error `unable to find face-name 'unifont Medium' in FontSet` occurs, it means that the old version of `unifont``is missing. The team of ``openstreetmap-carto` added as requirements the new and old version of unifont to load one of the two versions. So if you get an error like below, just ignore it :)

```
celeryworker_1    | Mapnik LOG> 2020-02-10 12:17:53: warning: unable to find face-name
↪'unifont Medium' in FontSet 'fontset-0'
celeryworker_1    | Mapnik LOG> 2020-02-10 12:17:53: warning: unable to find face-name
↪'unifont Medium' in FontSet 'fontset-1'
celeryworker_1    | Mapnik LOG> 2020-02-10 12:17:53: warning: unable to find face-name
↪'unifont Medium' in FontSet 'fontset-2'
```

19.5 How to delte just all django ohdm tables

To just delete django `OHDM` tables and not the other django tables like users use.:

```
$ docker-compose -f local.yml run --rm django python manage.py migrate ohdm zero
```

19.6 Cannot start service

When you try to start the containers and you get an error like:

```
ERROR: for postgres  Cannot start service postgres: Ports are not available: listen␣
↪tcp 127.0.0.1:5432: bind: Der Zugriff auf einen Socket war aufgrund der␣
↪Zugriffsrechte des Sockets unzulÄ¤ssig.
ERROR: Encountered errors while bringing up the project.
```

Then check if no other process is running on `5432`, `5555` and `8000`.

On linux & mac you can use:

```
$ netstat -vanp tcp | grep 5432
$ netstat -vanp tcp | grep 5555
$ netstat -vanp tcp | grep 8000
```

On Windows use CMD:

```
$ netstat -an
```

19.7 No such file or directory

When trying Docker on Windows on the first time, sometimes Windows will add `\r` on each file, but linux don't like it. If you get some errors like below, try to download the repo on a different way!:

```
/usr/bin/env: 'python\r': No such file or directory
```

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- modindex
- search

INDEX