Flask Meetup Data Scraper Documentation

Release 0.1

Steffen Exler

Feb 04, 2020

Contents

Table of Contents:

CHAPTER 1

Intro

Fulltext Meetup.com Search engine, download & index every meetup Group and the Group Events in a region every 28 days.

With the fulltext meetup search is it possible to search in every discription and any other field. So you know in wich places you done a talk.

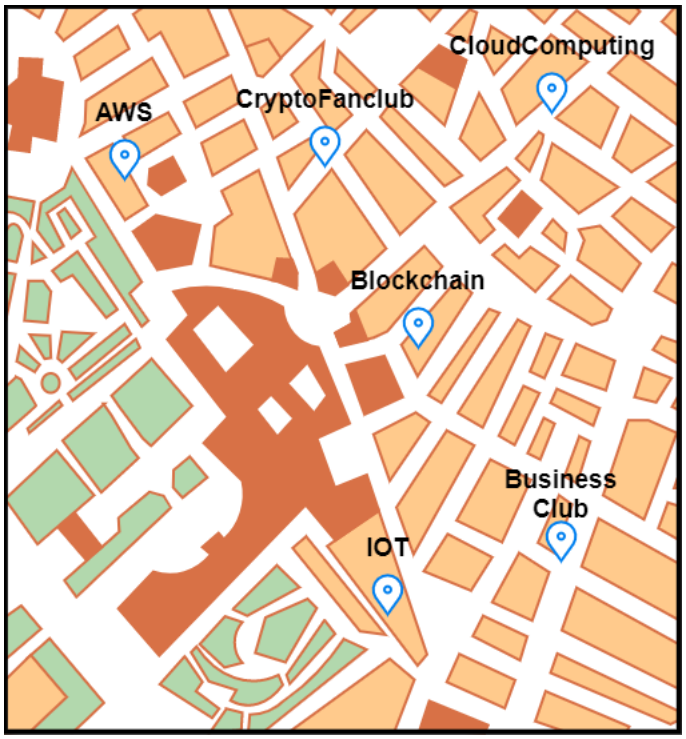

Fig. 1: Problem: In wich meetups did my team a talk?

Because it is not possible to create a fulltext search request on the official meetup API, so a way to solve the issue is

to download relevant meetup groups and index them into a elasticsearch.

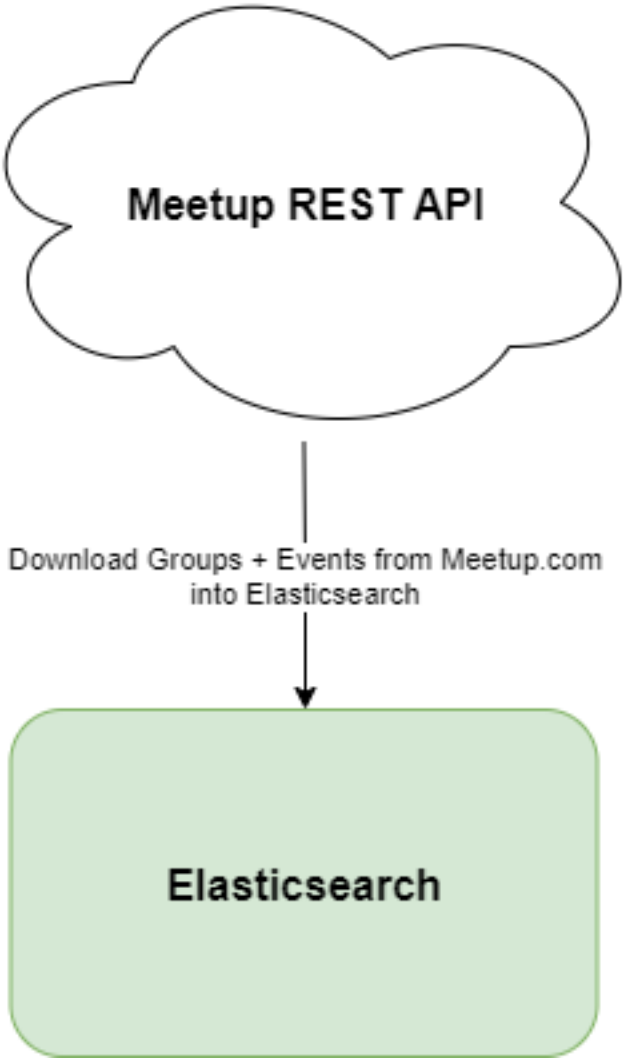

Fig. 2: Solution: Download every relevant group from meetup and index them into elasticsearch!

The Dataflow concept is that the API Server, wich is written in Python with the Flask webframework, download every 28 days all relevant meetup groups with there events and index them into elasticsearch. The search user use an angular app to communicate with the API server. For an easy deployment the angular app has it's own `NGINX` based docker container and the `traefik` container route every `http` & `https` traffik to the angular container expect `PUT` request. `Put` request are routet to the API server. Also `traefik` secure the communication with the enduser via SSL and used to handle basic auth request for the frontend & backend!

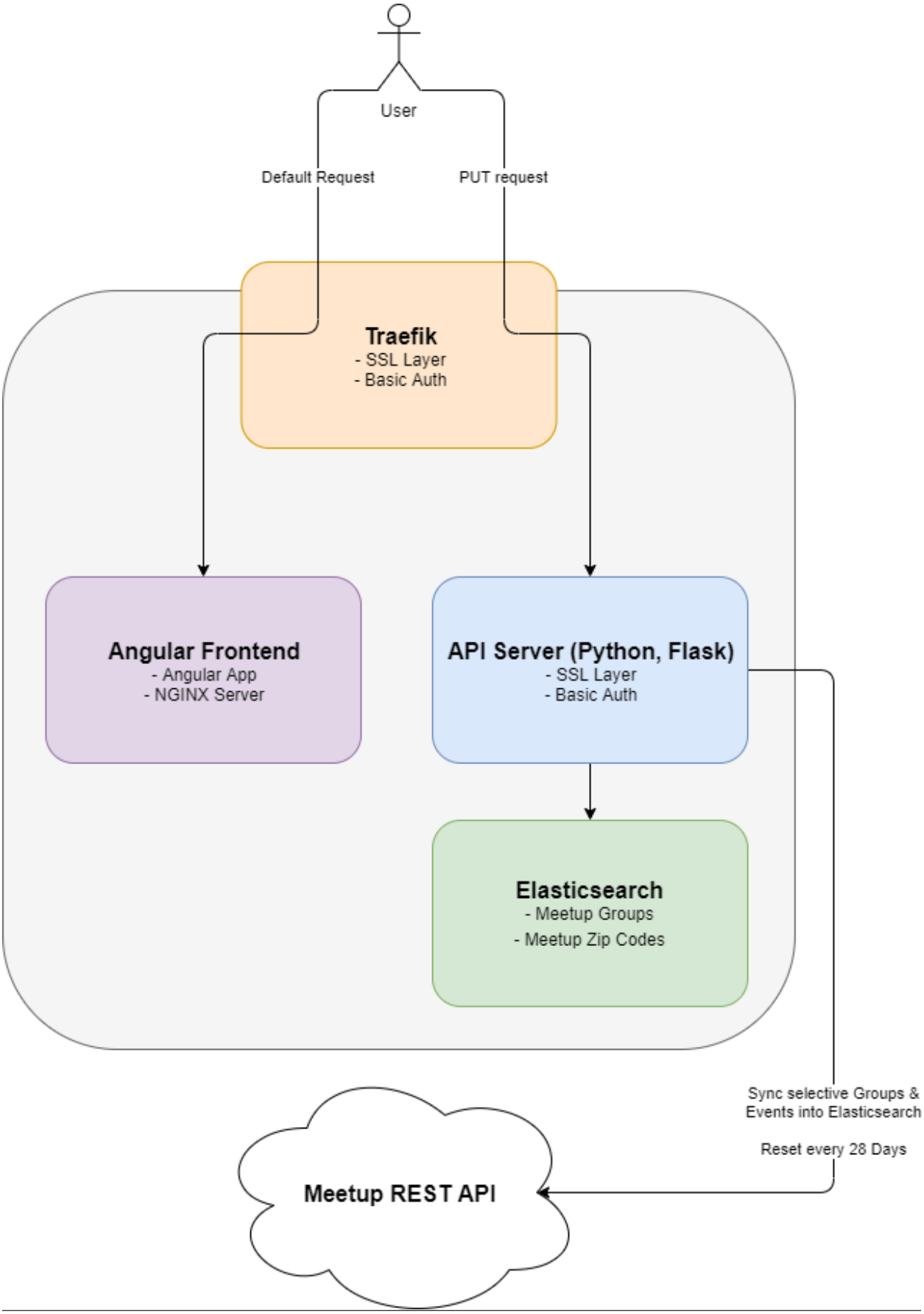

Fig. 3: DataFlow

CHAPTER 2

Getting started

Note: These instructions assume familiarity with Docker and Docker Compose.

2.1 Development & Production Version

The Project comes with 2 different Docker-Compose files wich are for development `local.yml` and production `production.yml`.

The development version start the website in debug mode and bind the local path `./` to the flask docker contaiers path `/app`.

For the production version, the docker container is build with the code inside of the container. Also the production version use redis as caching backend.

2.2 Quick install (Development Version)

Build the docker container.

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

Migrate all models into Elasticsearch

```
$ docker-compose -f local.yml run flask flask migrate_models
```

Load the Meetup Sandbox Group with all events.

```
$ docker-compose -f local.yml run flask flask get_group --sandbox True
```

Start the website.

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

Now the server is listen on http://localhost:5000 for any REST API requests.

2.3 Quick install (Production Version)

2.3.1 Settings

At first create the directory `./.envs/.production`

```
$ mkdir ./.envs/.production`
```

For flask container create a file `./.envs/.production/.flask` wich should look like:

To fill the section `# Meetup.com OAuth` you need an API account from Meetup.com. When setting up the meetup oauth account add `https://you-domain.com/callback` as your callback url & with `https://you-domain.com/login` you can log in with your meetup account.

To read how to handle a boundingbox in the section `# Meetup.com groups region` go to *load_zip_codes*.

For Elasticsearch container create a file `./.envs/.production/.elasticsearch` wich should look like below. For further information how to setup Elasticsearch with enviroment vars got to https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html

2.3.2 Add Users

Frontend & backend has the same endpoint for user authentification. Both use Basic Auth from traefik. To add a user, use `htpasswd` and store the user data into `compose/production/traefik/basic-auth-usersfile`. Example use in Linux:

```
$ sudo apt install apache2-utils # install htpasswd
$ htpasswd -c compose/production/traefik/basic-auth-usersfile username
```

2.3.3 Setup

Build the docker container.

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

Create the elasticsearch indexes.

```
$ docker-compose -f production.yml run flask flask migrate_models
```

Load Meetuup zip codes for a country.

```
$ docker-compose -f production.yml run flask flask load_zip_codes 47.2701114 55.
↪099161 5.8663153 15.0418087 # germany
$ docker-compose -f production.yml run flask flask load_zip_codes 45.817995 47.
↪8084648 5.9559113 10.4922941 # switzerland
$ docker-compose -f production.yml run flask flask load_zip_codes 46.3722761 49.
↪0205305 9.5307487 17.160776 # austria
```

Load Meetuup zip codes for a country.

```
$ docker-compose -f production.yml run flask flask load_zip_codes 47.2701114 55.
↪099161 5.8663153 15.0418087 # germany
$ docker-compose -f production.yml run flask flask load_zip_codes 45.817995 47.
↪8084648 5.9559113 10.4922941 # switzerland
$ docker-compose -f production.yml run flask flask load_zip_codes 46.3722761 49.
↪0205305 9.5307487 17.160776 # austria
```

Start the website.

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

2.3.4 Conjob

Add a cronjob to run every week. So that every `4` weeks the elasticsearch index should be resetet. If you want a another periode change the `4` with your periode time. But don't use a persiod over 30 days! It is forbidden by meetup.com!!:

```
0   3       *       *       6       docker-compose -f production.yml run flask flask␣
↪reset_index --reset_periode 4
```

Description what does this command do, is under *reset_index*.

CHAPTER 3

Usage Guide

Note: The usage guide is written for the development instance, to use the commands for prodcution change `-f local.yml` to `-f production.yml` for every command!

3.1 CLI

3.1.1 Help

For showing a general help for all commands run:

```
$ docker-compose -f local.yml run flask flask
```

To display a helptext for a single command, add to the command `--help` like:

```
$ docker-compose -f local.yml run flask flask shell --help
```

3.1.2 migrate_models

To use elasticsearch search, the index for the `groups` & `zip codes` musst be set. To do this run:

```
$ docker-compose -f local.yml run flask flask migrate_models
```

3.1.3 Load single group from Meetup.com

To download a single group from Meetup.com and index them into elasticsearch use `get_group`. For downloading a group use the `urlname`. When you are on a Meetup group page like https://www.meetup.com/de-DE/Meetup-API-Testing/, the urlname is in the browser URL field after the language-code block.

In the example for Meetup-API-Testing group, the language-code is de-DE and the urlname is Meetup-API-Testing.

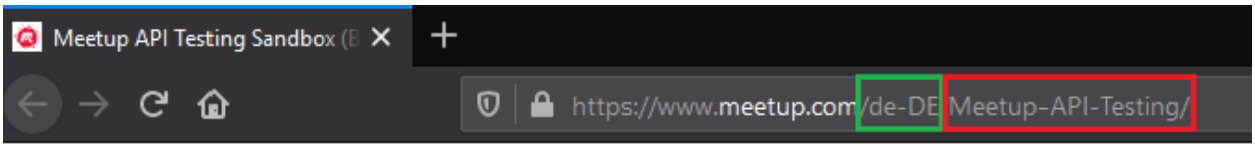

Fig. 1: Meetup group urlname screenshot

Also when you search for new groups in the meetup.com rest api, the `urlname` has it extra field for request a group entpoint.

So to download & index the Meetup-API-Testing group use:

```
$ docker-compose -f local.yml run flask flask get_group Meetup-API-Testing
```

The output should look like:

```
Starting flask-meetup-data-scraper_elasticsearch_1 ... done
Elasticsearch is available
Group Meetup API Testing Sandbox was updatet with 761 events
```

For testing purpose it's possible to load the sandbox group with the param `--sandbox True`, than it's not required to add a `urlname`.:

```
$ docker-compose -f local.yml run flask flask get_group --sandbox True
```

3.1.4 get_groups

Warning: The command `get_groups` is desiged for use only in development, for prodcution please use `load_groups`!

Load mutiple groups from JSON files. The JSON files muss include a key & a URL name. To download use the rest api direkt or via the meetup website https://secure.meetup.com/meetup_api/console/?path=/find/groups

An example Rest API request for the first 200 german groups -> https://api.meetup.com/find/groups?&sign=true&photo-host=public&country=DE&page=200&offset=0&only=urlname

After you downloaded the json, put them into `./meetup_data_scraper`. When you download the JSON's in a another directory set the path via `--json_path /app/your-dir/`. When you run the command in docker, you need to set the path inside the docker container.

```
$ docker-compose -f local.yml run flask flask get_groups
```

Example JSON file in `./compose/local/flask/meetup_groups/test-groups.json`

```
{
    "0": {
        "urlname": "Meetup-API-Testing"
    },
    "1": {
        "urlname": "None"
    },
```

(continues on next page)

(continued from previous page)

```
    "2": {
        "urlname": "connectedawareness-berlin"
    }
}
```

3.1.5 load_groups

Load all groups from every meetup.com zip code stored in elasticsearch use `load_groups`.

```
$ docker-compose -f local.yml run flask flask load_groups
```

`load_groups` accecpt 2 params, `--load_events False` will do **not** load all past events from a group (default is to load all past events) & `--country DE` will set to load only groups from a specific country by a country code. To get the meetup country codes check out meetup.com zip code.

```
$ docker-compose -f local.yml run flask flask load_groups --load_events False --
↪country DE
```

3.1.6 load_zip_codes

For downloading all meetup.com zip code use `load_zip_codes` with arguments of a boundingbox. A boundingbox is a gelocation area, to find out a boundingbox of a location like germany use the rest api of https://nominatim.openstreetmap.org/. To create a search query for nominatim add after `/search/` your query & after the query add `?format=json` for a json output. The full search request for germany is `https://nominatim.openstreetmap.org/search/germany?format=json`

Output:

```
[
    {
        "place_id": 235495355,
        "licence": "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/
↪copyright",
        "osm_type": "relation",
        "osm_id": 51477,
        "boundingbox": [
            "47.2701114",
            "55.099161",
            "5.8663153",
            "15.0418087"
        ],
        "lat": "51.0834196",
        "lon": "10.4234469",
        "display_name": "Deutschland",
        "class": "boundary",
        "type": "administrative",
        "importance": 0.8896814891722549,
        "icon": "https://nominatim.openstreetmap.org/images/mapicons/poi_boundary_
↪administrative.p.20.png"
    },
    ...
]
```

Now add the boundingbox to the `load_zip_codes`.

```
$ docker-compose -f local.yml run flask flask load_zip_codes 47.2701114 55.099161 5.
↪8663153 15.0418087 # germany
$ docker-compose -f local.yml run flask flask load_zip_codes 45.817995 47.8084648 5.
↪9559113 10.4922941 # switzerland
$ docker-compose -f local.yml run flask flask load_zip_codes 46.3722761 49.0205305 9.
↪5307487 17.160776 # austria
```

3.1.7 update_groups

To get all new past events from all groups in the elasticsearch use:

```
$ docker-compose -f local.yml run flask flask update_groups
```

3.1.8 reset_index

The `reset_index` command should only use when you want to delete your complete elasticsearch index & reload all groups from meetup.com. This should use as a cronjob at least once every 30 days!

```
$ docker-compose -f local.yml run flask flask reset_index
```

As param it possible to set the warning time in secounds with `--waring_time 30`, the default time span is 30 secounds.

```
$ docker-compose -f local.yml run flask flask reset_index --waring_time 30
```

Also you can add a weekly periode with `--reset_periode`. If this param is use, the command check if the current week is modulo by the value.

As an example to execute the command on only every 4 weeks use:

```
$ docker-compose -f local.yml run flask flask reset_index --reset_periode 4
```

The command check how many weeks are gone since 1970 (unix time) and calc them modulo `%` by for. So for the date `2020-01-30` is in the `2613` week since `1970-01-01`.:

```
2613 % 4 = 1
```

Since the rest of `2613 % 4` is not `0`, the command will exit. Only when the rest is `0` the command will be execute!

CHAPTER 4

Advanced topics

4.1 Changing Models

The elasticsearch models are stored in `meetup_search/models` and the tests are in `tests/models`. To edit the models read the Elasticsearch-DSL Docs.

4.2 Documentation

The docs are stored in `./docs` and written with Sphinx. The recommend way to host sphinx docs are with readthedocs.org.

To compile the docs as HTML use:

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

The html output goes to `docs/_build/html`

CHAPTER 5

IDE

This Project was created with Visual Studio Code and this section help you to setup your VS Code installation for this project.

5.1 Recommanded Extensions for VS-Code

5.1.1 Python

- Python
- MagicPython
- Visual Studio IntelliCode
- autoDocstring

5.1.2 Docker

- Docker
- Docker Linter

5.2 Install Python

Please use the same Version of Python as it used in the Flask Dockerfiles! Right now it is Python 3.8.

5.2.1 Windows 10

Note: Change the command `Python` to `py` when following the instructions!

To enable Python 3 in your Windows 10 Power please follow the article on Digitalocean.com

5.2.2 Mac

On mac you can use `brew`

```
$ brew install python3
```

5.2.3 Linux

In moste linux systems python is installed and maintaind out of the box, you just need to check if you use the same version as in in Dockerfiles.

5.3 Install Python dependencies

5.3.1 Virtualenv

If you like, you can install every dependency in a specific folder via virtualenv. To create a virtualenv for the project dependencies.

Virtualenv when Python 3 is the default python interpreter.

```
$ virtualenv venv
```

When you want to select a different python version use the param `-p`

```
$ virtualenv venv -p python38
```

To use the virtualenv use the `source` command.

```
$ source venv/bin/activate
```

5.3.2 Install development packages

Windows

```
$ python -m pip install -r .\requirements\local.txt
```

Mac / Linux

```
$ pip install -r requirements/local.txt
```

5.4 Code Format

This use Black to format this code, in VS Code you can set on every save to format the code in black. You can add auto format in black on every save when you add follow settings in your `settings.json`

```
{
    "editor.formatOnSave": true,
    "python.formatting.provider": "black",
    "editor.codeActionsOnSave": {
        "source.organizeImports": true
    },
}
```

To install `black` use `pip`.

For Windows:

```
$ python -m pip install black
```

For Mac / Linux:

```
$ pip install black
```

To format the code from the terminal you can use the black cli. For example to format the whole project use.

```
$ black ./
```

CHAPTER 6

Rest API Documentation

6.1 About

The REST API is based on Flask RESTful. To add or remove Endpoints modify in `app.py` the method `create_app`. Example how to add add suggestion & search:

```
# init flask api
api: Api = Api(app)
# add api endpoints
api.add_resource(MeetupSearchApi, "/")
api.add_resource(MeetupSearchSuggestApi, "/suggest/")
```

The code for the REST API is in `meetup_search/rest_api/api.py` and the tests are in `tests/rest_api/test_api.py`.

Also in `tests/rest_api/utily.py` are helper methods to tests the REST API!

6.2 Suggestion

`PUT /suggest/`

Return up to 5 suggestion based on group names.

Example, when send a `PUT /suggest/` with follow data:

```
{
    'query': 'jam',
}
```

The output will be like:

```
{"suggestions": [
    "Jam-Session Berlin",
    "Jam Time Amsterdam",
    "Jammy @ ROMA",
]}
```

6.3 Search

`PUT /`

Create a fulltext search on every group saved in Elasticsearch.

Example of every possible PUT Data:

```
{
    'query': 'my_query',
    'load_events': true,
    'page': 0,
    'limit': 10,
    'sort': ['urlname', '-meetup_id'],
    'geo_distance': '100km',
    'geo_lat': 52.13,
    'geo_lon': 13.12,
    'event_time_gte': '2019-11-01',
    'event_time_lte': '2020-01-01'
}
```

And the result will be a list of Group models, go to Meetup API Doc for fields description:

```
{
    'results': [
        # required fields
        'meetup_id': int,
        'urlname': str,
        'created': str,
        'description': str,
        'name': str,
        'link': str,
        'location': {
            'lat': float,
            'lon': float,
        },
        'members': int,
        'status': str,
        'timezone': str,
        'visibility': str,

        # extra optional fields from the rest api (get every venue location from all␣
↪events in the group)
        'venues': [
            'name': str,
            'location': {
                'lat': float,
                'lon': float,
            }
        ],
```

(continues on next page)

(continued from previous page)

```
        'venue_location_average': {
            'lat': float,
            'lon': float,
        },

        # optional fields
        'score': float;
        'nomination_acceptable': bool,
        'city': str,
        'city_link': str,
        'country': str,
        'fee_options_currencies_code': str,
        'fee_options_currencies_default': bool,
        'fee_options_type': str,
        'join_mode': str,
        'localized_country_name': str,
        'localized_location': str,
        'member_limit': int,
        'short_link': str,
        'state': str,
        'untranslated_city': str,
        'welcome_message': str,
        'who': str,
        'category_id': long,
        'category_name': str,
        'category_shortname': str,
        'category_sort_name': str,
        'meta_category_id': long,
        'meta_category_shortname': str,
        'meta_category_name': str,
        'meta_category_sort_name': str,
        'topics': [
            'meetup_id': str,
            'lang': str,
            'name': str,
            'urlkey': str,
        ],
        'organizer_id': int,
        'organizer_name': str,
        'organizer_bio': str,
        'events': [
            # required fields
            'meetup_id': str,
            'time': str,
            'name': str,
            'link': str,
            'date_in_series_pattern': bool,

            # optional fields
            'attendance_count': int,
            'attendance_sample': int,
            'attendee_sample': int,
            'created': str,
            'description': str,
            'duration': long,
            'fee_accepts': str,
            'fee_amount': int,
```

(continues on next page)

(continued from previous page)

```
            'fee_currency': str,
            'fee_description': str,
            'fee_label': str,
            'how_to_find_us': str,
            'status': str,
            'updated': str,
            'utc_offset': long,
            'venue_visibility': str,
            'visibility': str,
            'venue_address_1': str,
            'venue_address_2': str,
            'venue_address_3': str,
            'venue_city': str,
            'venue_country': str,
            'venue_localized_country_name': str,
            'venue_name': str,
            'venue_phone': str,
            'venue_zip_code': str,
            'venue_location': {
                'lat': float,
                'lon': float,
            },
            'event_host_host_count': int,
            'event_host_id': int,
            'event_host_intro': str,
            'event_host_join_date': str,
            'event_host_name': str,
        ]
    ],
    'hits': int,
    'map_center': {
        'lat': float,
        'lon': float,
    }
}
```

6.3.1 PUT Data fields

query

`query` is the only `required` field for a search request. The query has to be a string and could also use wildcards like `*`. Example for a minimal search request:

```
{
    'query': 'my_query',
}
```

load_events

By default events will not be send through a search request, only if `load_events` is set to `True`:

```
{
    'load_events': true,
}
```

pagination

For pagination use the fields `limit` (how many groups will load on a request) and `page`.

When not set the default value for `page` is `0` and for `limit` is it `10`.

`limit` only accept `5`, `10`, `25`, `100` as valid value!

It's possible to just use `page` or `limit` without the other, than the default values will be used!

Example for the secound page with 25 entries per page.:

```
{
    'query': 'my_query',
    'page': 2,
    'limit': 55,
}
```

sorting

It's possible to sort the groups by field (only work on group fields, not an nested fields like `events` or `topic`).

To costimize sorting read the sort docs!

To sort a query by `urlname` in `asc` and `meetup_id` in `desc` use:

```
{
    'query': 'my_query',
    'sort': ['urlname', '-meetup_id'],
}
```

geo_distance

To filter groups by a geo_distance the fields `geo_distance`, `geo_lat` & `geo_lon` have to be all set, there is no default value!

The distance filter check for events venue location, if a group has any event with a venue in the distance it will be return.

`geo_distance` accept elasticsearch distance units

`geo_lat` & `geo_lon` accecpt float values. To get geopoints of citys and points of intresst you can use Nominatim.

For deeper explination go to Geo-distance query doc

Example for a distance request on Berlin with 100km:

```
{
    'query': 'my_query',
    'geo_distance': '100km',
    'geo_lat': 52.520008,
    'geo_lon': 13.404954,
}
```

Filter by event time

The fields `event_time_gte` and `event_time_lte` are used to filter events by the time when they was done.

Attation, when at leats one event of a group was hit, the hole group with all events will be returned!

To filter events with a date greater or equal date than `2019-11-01` use:

```
{
    'query': 'my_query',
    'event_time_gte': '2019-11-01',
}
```

To filter events with a date less or equal than `2020-01-01` use:

```
{
    'query': 'my_query',
    'event_time_lte': '2020-01-01'
}
```

It's alo possible to use both filter at once, so to filter a date greater or equal date than `2019-11-01` and less or equal than `2020-01-01` use:

```
{
    'query': 'my_query',
    'event_time_gte': '2019-11-01',
    'event_time_lte': '2020-01-01'
}
```

CHAPTER 7

C/I

7.1 About

Flask Meetup Data Scraper use Github.com Marketplace Apps to maintain the project. Every App is for free for Open Source projects!

7.2 Code Style

7.2.1 Black

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

7.3 Tests

7.3.1 Travis

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

Travis config is `.travis.yml`

7.4 Documentation

7.4.1 Readthedocs.org

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

Travis config is `.readthedocs.yml`

7.5 Code Review

7.5.1 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.

7.5.2 DeepSource.io

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

DeepSource config is `.deepsource.toml`

7.6 Dependencies

7.6.1 Pyup.io

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

Pyup config is `.pyup.yml`

7.6.2 Dependabot.com

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

Dependabot config is `.dependabot/config.yml`

CHAPTER 8

Elasticsearch Queries

The main Elasticsearch Query is written in `meetup_search/rest_api/api.py` and the tests are in `tests/rest_api/test_api.py`. This project use https://github.com/elastic/elasticsearch-dsl-py to handle Elasticsearch, if you want to modify the query, go to https://elasticsearch-dsl.readthedocs.io/en/latest/ for help.

To run the tests for the `api` run:

```
$ docker-compose -f local.yml run flask coverage run -m pytest -s tests/rest_api/test_
↪api.py
```

To run a single test use:

```
$ docker-compose -f local.yml run flask coverage run -m pytest -s tests/rest_api/test_
↪api.py::test_search_query
```

CHAPTER 9

Frontend

The Frontend is written as an Angular app. The source code is in a extra git repo https://github.com/linuxluigi/meetup-data-scraper-angular

For developing the frontend it's best to run the flask app in background. The develoing settings of the app try to make `PUT` request on `http://localhost:5000` and the production site is designt to run on the same domain as the backend.

To run the frontend & backend on the same domain traefik is setup to handle every `http` & `https` request. The default setup is that every traffik goes to the angular app (NGINX server) and only http `PUT` request go to the flask backend app.

Fig. 1: The Landingpage from the angular frontend

CHAPTER 10

Testing

Warning: Do not run Tests on Production Systems!!! Tests will destroy your Elasticsearch Index!!!

10.1 Pytest

The unit test are stored in the `tests` folder and written with the Pytest Framework.

10.2 Conftest

When executing a test like.

```
$ docker-compose -f local.yml run flask coverage run -m pytest
```

Pytest will at first go to the file `conftest.py` and execute the method `pytest_runtest_setup` bevor each test and after each test the method `pytest_runtest_teardown` will be executed.

Every method witch is an annotation `@pytest.fixture` can be used in any test method as a param, for deteaild explination go to https://docs.pytest.org/en/latest/fixture.html

10.3 Create new test

To create a new test just create a python file with the prefix `test_` in the folder `/tests`. It also possible to bundle test in a python package, for that create a folder in `/tests` and add a empty file `__init__.py` in the new folder, so that python recognize the folder as a python package.

In the new test file with the prefix `tests_` (example: `/tests/test_user.py`) add method also with the prefix `test_`. An example for the `/tests/test_user.py` test file would look like:

```
def test_user():
    assert get_user(username="Hugo").username == "Hugo"
```

10.4 Execute tests

To execute all tests run:

```
$ docker-compose -f local.yml run flask coverage run -m pytest
```

To run all tests in a specific module add the path like, in the example run all test in the path `tests/api_client`:

```
$ docker-compose -f local.yml run flask coverage run -m pytest tests/api_client
```

For running all test in a file, just add the full path of the file:

```
$ docker-compose -f local.yml run flask coverage run -m pytest tests/api_client/test_
↪json_parser.py
```

To run a single method add 2 colons to the file path with a method name.:

```
$ docker-compose -f local.yml run flask coverage run -m pytest tests/api_client/test_
↪json_parser.py::test_get_group_from_response
```

Note: If you add `-s` as param when executing a tests, you can see the terminal output from the test. `docker-compose -f local.yml run flask coverage run -m pytest -s`

CHAPTER 11

Troubleshooting

This page contains some advice about errors and problems commonly encountered during the development of Meetup Data Scraper.

11.1 max virtual memory areas vm.max_map_count [65530] likely too low, increase to at least [262144]

When using docker on some machines, you will need to manually extend the max virtual memory. For CentOS & Ubuntu use:

```
$ sudo sysctl -w vm.max_map_count=262144
```

Or add it permanently use:

```
$ echo "vm.max_map_count=262144" | sudo tee -a /etc/sysctl.conf
$ sudo reboot
```

For more detils go to -> https://www.elastic.co/guide/en/elasticsearch/reference/current/vm-max-map-count.html

11.2 Build faild -> out of memory

Building need quit a lot of RAM, if container like elasticsearch run in background you can ran out of memory. So you need to stop all all containers.:

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

11.3 Error when starting container under Windows

> `ERROR`: for flask-meetup-data-scraper_flask_1 Cannot start service flask: error while creating mount source path `/host_mnt/c/Users/.../dev/flask-meetup-data-scraper: mkdir /host_mnt/c: file exists`

When the error comes, in most cases is it enought to just delete all unused container.

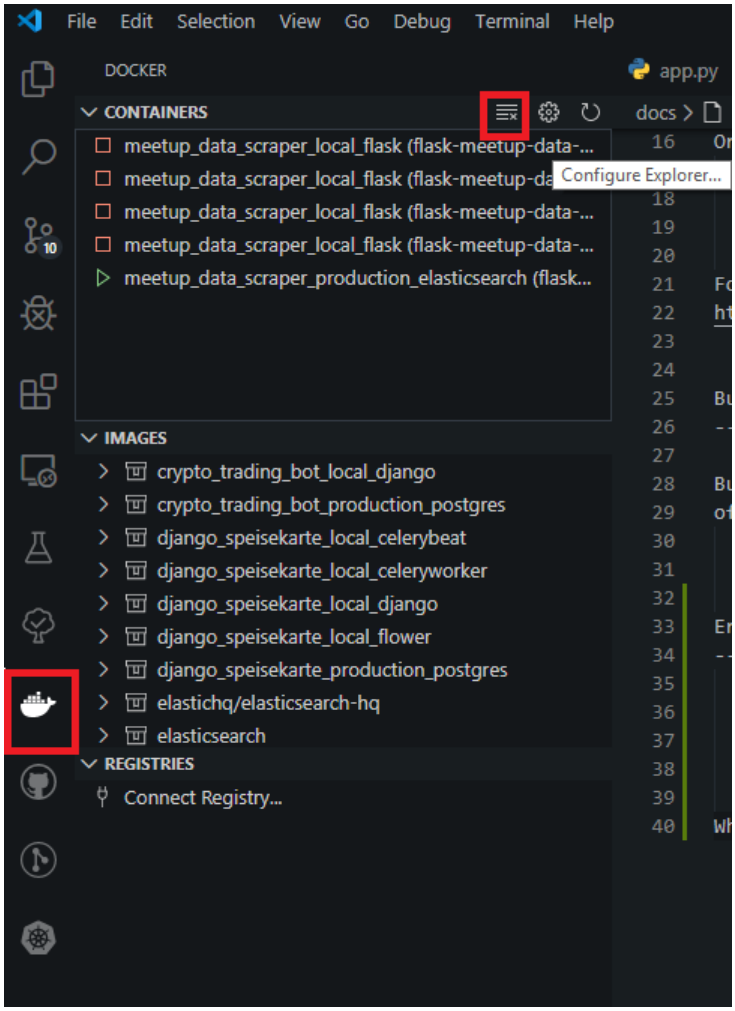

Fig. 1: Delte unused docker containers

CHAPTER 12

FAQ

12.1 What are the minimum hardware requirements?

To host it with docker you will need at leat a vServer with 2GB RAM, 10GB disk space & 1 CPU.

12.2 How to set the domain for a production site?

Replace in `compose\production\traefik\traefik.yml` the entry `meetup-data-scraper.de` with your target domain.

CHAPTER 13

Indices & Tables

- genindex
- modindex
- search

Index