Luftdatenpumpe

The Earth Observations Developers

Dec 19, 2022

CONTENTS:

CHAPTER

ONE

ABOUT

Acquire and process live and historical air quality data without efforts.

1.1 Description

Filter by station-id, sensor-id and sensor-type, apply reverse geocoding, store into time-series and RDBMS databases (InfluxDB and PostGIS), publish to MQTT, output as JSON, or visualize in Grafana. Data sources: Sensor.Community (luftdaten.info), IRCELINE, and OpenAQ.

The *Luftdatenpumpe README* has more details about features, screenshots, and basic usage information.

1.2 Example

Visualization of particulate matter pollution on New Year's Eve 2018, as seen through measurement data of the LDI network. More information at Animation of fine dust pollution on New Year's Eve 2018 across Europe.

Fig. 1: Particulate matter pollution on New Year's Eve 2018 in Europe/Germany.

CHAPTER

TWO

HANDBOOK

2.1 Luftdatenpumpe

Acquire and process live and historical air quality data without efforts.

- **Status**

- **Usage**

- **Compatibility**

2.1.1 About

Acquire and process live and historical air quality data without efforts.

Filter by station-id, sensor-id and sensor-type, apply reverse geocoding, store into time-series and RDBMS databases (InfluxDB and PostGIS), publish to MQTT, output as JSON, or visualize in Grafana.

Data sources: Sensor.Community (luftdaten.info), IRCELINE, and OpenAQ.

2.1.2 Features

1. Luftdatenpumpe acquires the measurement readings either from the livedata API of luftdaten.info or from its archived CSV files published to *archive.luftdaten.info*. To minimize impact on the upstream servers, all data gets reasonably cached.
2. While iterating the readings, it optionally filters on station-id, sensor-id or sensor-type and restrains information processing to the corresponding stations and sensors.
3. Then, each station's location information gets enhanced by
   - attaching its geospatial position as a Geohash.
   - attaching a synthetic real-world address resolved using the reverse geocoding service Nominatim by OpenStreetMap.
4. Information about stations can be
   - displayed on STDOUT or STDERR in JSON format.
   - filtered and transformed interactively through jq, the swiss army knife of JSON manipulation.
   - stored into RDBMS databases like PostgreSQL using the fine dataset package. Being built on top of SQLAlchemy, this supports all major databases.
   - queried using advanced geospatial features when running PostGIS, please follow up reading the Luftdatenpumpe PostGIS tutorial.
5. Measurement readings can be
   - displayed on STDOUT or STDERR in JSON format, which allows for piping into jq again.
   - forwarded to MQTT.
   - stored to InfluxDB and then
   - displayed in Grafana.

2.1.3 Synopsis

```
# List networks
luftdatenpumpe networks

# List LDI stations
luftdatenpumpe stations --network=ldi --station=49,1033 --reverse-geocode

# Store list of LDI stations and metadata into RDBMS database (PostgreSQL), also display␣
```

(continues on next page)

(continued from previous page)

```
↪on STDERR
luftdatenpumpe stations --network=ldi --station=49,1033 --reverse-geocode --
↪target=postgresql://luftdatenpumpe@localhost/weatherbase

# Store LDI readings into InfluxDB
luftdatenpumpe readings --network=ldi --station=49,1033 --target=influxdb://
↪luftdatenpumpe@localhost/luftdaten_info

# Forward LDI readings to MQTT
luftdatenpumpe readings --network=ldi --station=49,1033 --target=mqtt://mqtt.example.org/
↪luftdaten.info
```

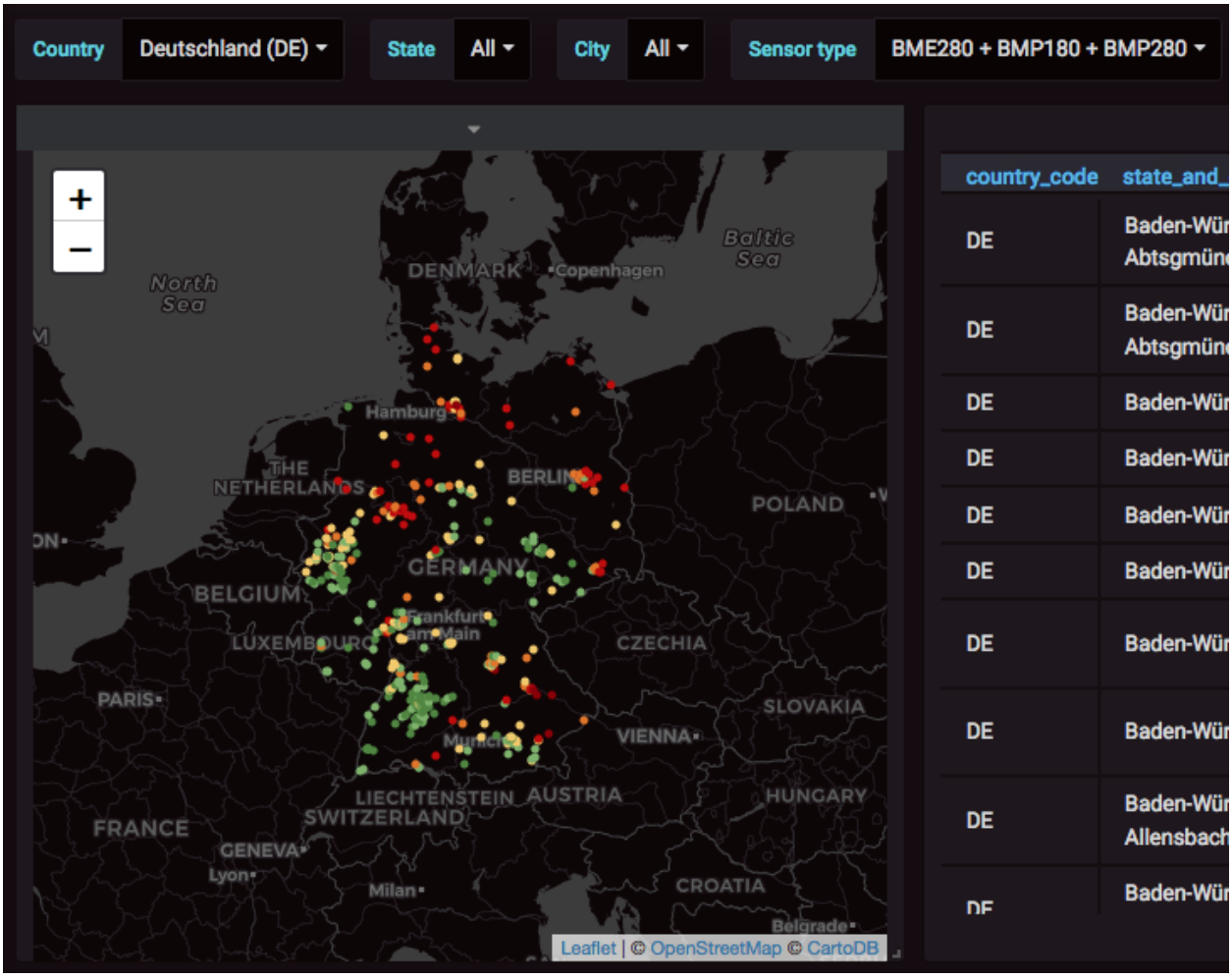
For a full overview about all program options including meaningful examples, you might just want to run `luftdatenpumpe --help` on your command line, or visit the Luftdatenpumpe usage documentation section.

2.1.4 Screenshots

Luftdaten-Viewer displays stations and measurements from luftdaten.info (LDI) in Grafana.

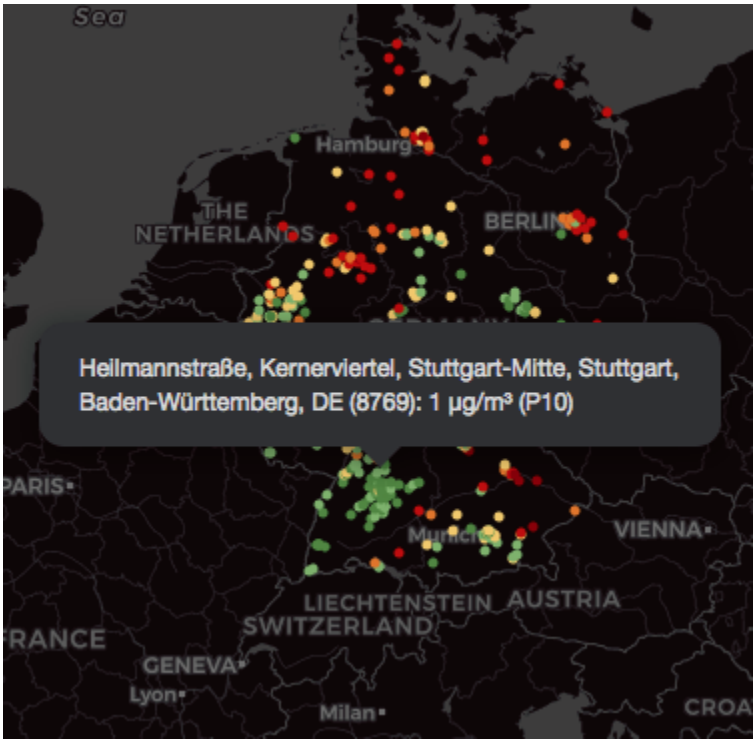
Map display and filtering

- Filter by different synthesized address components and sensor type.
- Display measurements from filtered stations on Panodata Map Panel.
- Display filtered list of stations with corresponding information in tabular form.
- Measurement values are held against configured thresholds so points are colored appropriately.



Map popup labels

- Humanized label computed from synthesized OpenStreetMap address.
- Numeric station identifier.
- Measurement value, unit and field name.



2.1.5 Installation

If you are running Python 3 already, you can installing the program using `pip`. We recommend to use a Python virtualenv.

```
pip install luftdatenpumpe --upgrade
```

At this point, you should be able to conduct simple tests like `luftdatenpumpe stations` as seen in the synopsis section above. At least, you should verify the installation succeeded by running:

```
luftdatenpumpe --version
```

At install Luftdatenpumpe, you will find more detailed installation instructions about how to install and configure auxiliary services, and eventually resolve some prerequisites.

2.1.6 Luftdaten-Viewer

About

Using Luftdatenpumpe, you can build user-friendly interactive GIS systems on top of PostGIS, InfluxDB and Grafana. This setup is called "Luftdaten-Viewer", and some example scenarios can be inspected at Luftdatenpumpe gallery.

Instructions

These installation instructions outline how to setup the whole system to build similar interactive data visualization compositions of map-, graph- and other panel-widgets like outlined in the "Testimonials" section.

- Luftdaten-Viewer Applications
- Luftdaten-Viewer Databases
- Luftdaten-Viewer Grafana

2.1.7 Other projects

Sensor.Community public data aggregator

Visualize recent sensor data on a world map for Sensor.Community and for different other official networks, like EEA, Luchtmeetnet, Atmo AURA/Sud/Occitanie, and Umweltbundesamt.

- https://github.com/pjgueno/SCPublicData
- https://forum.sensor.community/t/scraping-pm-data-help-needed/1448

2.1.8 Project information

Contributions

Any kind of contribution, feedback, or patch, is much welcome. Create an issue or submit a patch if you think we should include a new feature, or to report or fix a bug.

Resources

- Source code
- Documentation
- Community Forum
- Python Package Index (PyPI)

License

The project is licensed under the terms of the GNU AGPL license, see LICENSE.

Content attributions

The copyright of particular images and pictograms are held by their respective owners, unless otherwise noted.

- Water Pump Free Icon from Icon Fonts is licensed by CC BY 3.0.

2.2 Setup

The whole system is based on Grafana, InfluxDB, PostGIS, and Redis. This section of the documentation will guide you through all steps to setup and configure/provision the corresponding services.

2.2.1 Installation and configuration

Install Luftdatenpumpe

Setup

Install prerequisites:

```
apt install build-essential python3-dev libicu-dev
```

Install Luftdatenpumpe:

```
pip install luftdatenpumpe
```

Note:

- We recommend to use a *Python virtualenv* to install and operate this software independently from your local system-wide Python installation.
- In order to make the `luftdatenpumpe` command available system-wide, just place a symlink into `/usr/local/bin`, like:

  ```
  ln -s /opt/luftdatenpumpe/.venv/bin/luftdatenpumpe /usr/local/bin/luftdatenpumpe
  ```

Troubleshooting

Note: `luftdatenpumpe` depends on the PyICU package. Sometimes, `pkg-config` is not able to find the appropriate ICU installation, like:

```
RuntimeError:
Please set the ICU_VERSION environment variable to the version of
ICU you have installed.
```

So, you might try to do things like:

```
# Announce path to icu4c.
$ export PKG_CONFIG_PATH="/usr/local/opt/icu4c/lib/pkgconfig"

# To verify if that works, run:
$ pkg-config --modversion icu-i18n
```

Luftdaten-Viewer Applications

This section of the documentation outlines how to install the prerequisites for a full system based on Luftdatenpumpe. It is Grafana, InfluxDB, PostGIS, and Redis.

Debian

CentOS

macOS

Configure package repository

Hiveeyes is hosting recent releases of InfluxDB and Grafana there. We are mostly also running exactly these releases on our production servers.

Add Hiveeyes package repository:

```
wget -qO - https://packages.hiveeyes.org/hiveeyes/foss/debian/pubkey.txt | apt-key add -
```

Add Hiveeyes package repository, e.g. by appending this to `/etc/apt/sources.list`:

```
deb https://packages.hiveeyes.org/hiveeyes/foss/debian/ testing main foundation
```

Reindex package database:

```
apt install apt-transport-https
apt update
```

Install packages

```
apt install influxdb postgis redis-server redis-tools grafana
```

Todo: Transfer from `belgiumAir/installation.rst`.

```
brew install grafana influxdb postgis redis
```

Luftdaten-Viewer Databases

Introduction

This section of the documentation outlines how to provision the PostGIS and InfluxDB databases. It will assume all services are properly installed and configured on your system, and otherwise will give you instructions how to start the corresponding services in sandbox mode.

Prerequisites

When running in sandbox mode, those commands will start the services required to follow this tutorial. It is InfluxDB, PostGIS, and Redis:

```
make influxdb-start
make postgis-start
make redis-start
```

When running in production mode, you may need to configure your services to provide convenient authentication. On this matter, please have a look at *Notes about PostgreSQL authentication*.

PostGIS

Create and provision PostGIS database

Connect to PostGIS:

```
psql postgres://postgres@localhost:5432
```

When aiming to connect to PostGIS on a classic Linux host, where PostGIS is installed as a system service, those commands might work better:

```
su - postgres
psql
```

Create database:

```
CREATE DATABASE weatherbase;
\connect weatherbase;
```

Enable PostGIS extension:

```
CREATE EXTENSION postgis;
```

Create users:

```
CREATE ROLE luftdatenpumpe WITH LOGIN;
CREATE ROLE grafana WITH LOGIN PASSWORD 'readonly';
\q
```

Import data

Pre-flight checks:

```
psql postgres://luftdatenpumpe@localhost:5432/weatherbase
```

Run `luftdatenpumpe` for the first time to manifest the database schema.

LDI

IRCELINE

```
luftdatenpumpe stations \
    --network=ldi --station="49,1033" --reverse-geocode \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase --progress
```

```
luftdatenpumpe stations \
    --network=irceline --station="1030,1751" --reverse-geocode \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase --progress
```

Create database view and grant permissions to “grafana” user.

LDI

IRCELINE

```
luftdatenpumpe database --network=ldi \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase \
    --create-view --grant-user=grafana
```

```
luftdatenpumpe database --network=irceline \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase \
    --create-view --grant-user=grafana
```

Note: These steps will have to be performed **in order** as the last `--create-view` step will only work after data in the tables has been materialized.

Sanity checks

Let’s have a look if everything worked.

Database schema

As visible by an administrator.

```
psql -U luftdatenpumpe -h localhost -d weatherbase --command '\dtv ldi_*'

               List of relations
 Schema |     Name     | Type  |     Owner
--------+--------------+-------+----------------
 public | ldi_network  | view  | luftdatenpumpe
 public | ldi_osmdata  | table | luftdatenpumpe
 public | ldi_sensors  | table | luftdatenpumpe
 public | ldi_stations | table | luftdatenpumpe
(4 rows)
```

Data

- Query the database view `ldi_network` here.
- Use read-only account pretending to be Grafana.

```
psql \
    --username=grafana --host=localhost \
    --dbname=weatherbase --command='SELECT COUNT(*) FROM ldi_network;'
```

```
 count
-------
  1391
```

InfluxDB

Create and provision InfluxDB database

```
luftdatenpumpe readings --network=ldi --station="49,1033" \
    --target=influxdb://luftdatenpumpe@localhost/luftdaten_info
```

Sanity checks

Let's have a look if everything worked.

Database schema

```
influx \
    -host localhost -username luftdatenpumpe \
    -database luftdaten_info \
    -execute 'SHOW FIELD KEYS; SHOW TAG KEYS;'
```

```
fieldKey    fieldType
--------    ---------
P1          float
P2          float
humidity    float
temperature float

tagKey
------
geohash
sensor_id
station_id
```

Database content

```
influx \
    -host localhost -username luftdatenpumpe \
    -database luftdaten_info \
    -execute 'SHOW TAG VALUES WITH KEY = station_id;'
```

```
key        value
---        -----
station_id 1071
station_id 28
```

```
influx \
    -host localhost -username luftdatenpumpe \
    -database luftdaten_info \
    -execute 'SELECT COUNT(*) FROM ldi_readings;'
```

```
time count_P1 count_P2 count_humidity count_temperature
---- -------- -------- -------------- -----------------
0    4        4        4              4
```

Luftdaten-Viewer Grafana

This walkthrough brings everything in place to visualize the data feed of Sensor.Community, formerly `luftdaten.info`, using Grafana.

Installation

Flux datasource

Flux datasource:

```
grafana-cli plugins install grafana-influxdb-flux-datasource
```

Panodata Map Panel

The map component for Grafana used within this setup is the Panodata Map Panel.

Because it is not a built-in plugin, and not available on the Grafana plugin catalog, it is not signed. To allow using unsigned plugins starting with Grafana 7.x, please use the allow_loading_unsigned_plugins configuration setting, which is located within the `[plugins]` section of `/etc/grafana/grafana.ini`.

```
allow_loading_unsigned_plugins = panodata-map-panel
```

Install Panodata Map Panel:

```
grafana-cli \
    --pluginUrl https://github.com/panodata/panodata-map-panel/releases/download/0.16.0/
```

(continues on next page)

(continued from previous page)

```
↪panodata-map-panel-0.16.0.zip \
   plugins install panodata-map-panel
```

Finally, restart your Grafana instance:

```
systemctl restart grafana-server
```

Warning: Please note this procedure has been confirmed to work with Grafana versions up to Grafana 8.x. It has not been verified on the most recent Grafana 9.x version.

LDI Metadata

Storage for station list

The Panodata Map Panel (ex. Grafana Worldmap Panel) requires a JSON file `ldi-stations.json` to display appropriate popup labels to each data point on the map.

This section of the documentation describes how to create the corresponding file using `luftdatenpumpe` in a way it can be picked up by the map panel through HTTP.

Create directory for stations file:

```
mkdir -p /var/lib/grafana/data/json
chown -R grafana:grafana /var/lib/grafana/data
ln -sf /var/lib/grafana/data /usr/share/grafana/public/
```

Let all processing happen on network "LDI":

```
export LDP_NETWORK=ldi
```

Define stations file:

```
stationsfile=/var/lib/grafana/data/json/ldi-stations.json
```

Note: On a macOS/Homebrew system, Grafana is installed to `/usr/local/share/grafana` and `/usr/local/var/lib/grafana`. You might have to adjust the `$stationsfile` path to the environment you are running this program on:

```
mkdir -p /usr/local/var/lib/grafana/data/json
ln -sf /usr/local/var/lib/grafana/data /usr/local/share/grafana/public/

stationsfile=/usr/local/var/lib/grafana/data/json/ldi-stations.json
```

Create station list

Write stations and metadata to RDBMS database (PostgreSQL):

```
# Acquire baseline from live API.
luftdatenpumpe stations --reverse-geocode \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase --progress

# Optionally acquire more stations from CSV archive files.
luftdatenpumpe stations --reverse-geocode \
    --source=file:///var/spool/archive.luftdaten.info \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase --progress
```

Create RDBMS database view `ldi_network`:

```
# This is important to do *after* importing the metadata, as the
# previous steps will properly populate the database on the first hand.
luftdatenpumpe database \
    --target=postgresql://luftdatenpumpe@localhost/weatherbase --create-views --grant-
↪user=grafana
```

Export station metadata from RDBMS database (PostgreSQL) to JSON file for Panodata Map Panel:

```
luftdatenpumpe stations \
    --source=postgresql://luftdatenpumpe@localhost/weatherbase \
    --target=json.flex+stream://sys.stdout \
    --target-fieldmap='key=station_id|str,name=road_and_name_and_id' > $stationsfile
```

Check:

```
export GRAFANA_URL=http://localhost:3000
http $GRAFANA_URL/public/data/json/ldi-stations.json | jq length
760
```

LDI Grafana artefacts

Prerequisites

```
# Define the URL to your Grafana instance.
# This saves you from having to supply it all over again to the subsequent commands.
export GRAFANA_URL=http://localhost:3000

# Sign in to your Grafana instance once.
# This saves you from having to supply "--auth=admin:admin" on every subsequent␣
↪invocation.
http --session=grafana $GRAFANA_URL --auth=admin:admin
```

Note: When running on localhost, use this URL instead:

```
export GRAFANA_URL=http://localhost:3000
```

Datasources

```
# Create data source object for "weatherbase @ PostgreSQL".
luftdatenpumpe grafana --kind=datasource --name=weatherbase \
    | http --session=grafana POST $GRAFANA_URL/api/datasources

# Create data source object for "luftdaten_info @ InfluxDB".
luftdatenpumpe grafana --kind=datasource --name=influxdb \
    --variables=tsdbDatasource=luftdaten_info \
    | http --session=grafana POST $GRAFANA_URL/api/datasources
```

Note: Before being able to create the data source objects again, you will have to delete them first:

```
http --session=grafana DELETE $GRAFANA_URL/api/datasources/name/weatherbase
http --session=grafana DELETE $GRAFANA_URL/api/datasources/name/luftdaten_info
```

Dashboards

Create dashboard with graph panel:

```
luftdatenpumpe grafana --kind=dashboard --name=trend \
    --variables=tsdbDatasource=luftdaten_info,sensorNetwork=ldi \
    --fields=pm2-5=P2,pm10=P1 \
    | http --session=grafana POST $GRAFANA_URL/api/dashboards/db
```

Create dashboard with map and table panels:

```
luftdatenpumpe grafana --kind=dashboard --name=map \
    --variables=tsdbDatasource=luftdaten_info,sensorNetwork=ldi,jsonUrl=/public/data/
↪json/ldi-stations.json,autoPanLabels=false \
    --fields=pm2-5=P2,pm10=P1 \
    | http --session=grafana POST $GRAFANA_URL/api/dashboards/db
```

Note: This references the station list JSON file created in one of the previous steps.

2.2.2 Options

Luftdaten-Viewer Cron Job

Setup

Create “workbench” user:

```
useradd --shell=/bin/bash --create-home workbench
```

Install cron file:

```
curl --silent https://raw.githubusercontent.com/earthobservations/luftdatenpumpe/main/
↪etc/luftdaten-viewer.cron > /etc/cron.d/luftdaten-viewer
curl --silent https://raw.githubusercontent.com/earthobservations/luftdatenpumpe/main/
↪tools/pflock > /usr/local/bin/pflock
curl --silent https://raw.githubusercontent.com/earthobservations/luftdatenpumpe/main/
↪tools/safewrite > /usr/local/bin/safewrite
chmod +x /usr/local/bin/pflock /usr/local/bin/safewrite
```

Notes

pflock

Please modify `/usr/local/bin/pflock` on CentOS systems as `/var/lock` does not yield appropriate permissions by default to use the `/tmp` directory:

```
/tmp/program-${name}.pflock
```

Data acquisition rate

Also, you might want to increase the data acquisition rate in `/etc/cron.d/luftdaten-viewer`:

```
# Run data import each 5 minutes
*/5     *    * * *    workbench       pflock luftdatenpumpe readings --country=BE --target=
↪${tsdb_uri} >/dev/null 2>&1
```

Luftdaten-Viewer data loss monitoring

Install

Install “monitoring-check-grafana”.

Monitor a Grafana datasource against data becoming stale to detect data loss or other dropout conditions.

– https://github.com/daq-tools/monitoring-check-grafana

Configure

Introduction

Use this Icinga configuration object to monitor your data source. You will need to find out about your Grafana data source index. Here, it is “10”, see `https://daq.example.org/grafana/api/datasources/proxy/10/query`.

It has probably on the output of the command when creating the respective data source in Grafana:

```
# Create data source object for "luftdaten_info @ InfluxDB".
luftdatenpumpe grafana --kind=datasource --name=luftdaten_info \
    | http --session=grafana POST $GRAFANA_URL/api/datasources
```

Icinga configuration object

```
// Monitor "Luftdaten-Viewer" data feed for data loss
object Service "Luftdaten-Viewer data freshness" {
  import "baseline-service"

  check_command         = "check-grafana-datasource-stale"

  host_name             = "slartibartfast.example.org"
  vars.sla              = "24x7"

  vars.grafana_uri      = "https://daq.example.org/grafana/api/datasources/proxy/10/query
↪"
  vars.grafana_database = "luftdaten_info"
  vars.grafana_table    = "ldi_readings"
  vars.grafana_warning  = "20m"
  vars.grafana_critical = "1d"

  vars.notification.mail.groups = [ ]
  vars.notification.mail.users  = [ "john-doe", "max-mustermann" ]

}
```

Luftdaten-Viewer Grafana for VMM

This walkthrough builds upon the baseline documentation about setting up Luftdatenpumpe with Grafana, and is a guideline for VMM to visualize the data feed of IRCELINE.

Setup

Let all processing happen on network “IRCELINE”:

```
export LDP_NETWORK=irceline
```

LDI network

Create dashboard with map and table panels for LDI Belgium:

```
luftdatenpumpe grafana --kind=dashboard --name=map \
    --variables=tsdbDatasource=luftdaten_info,sensorNetwork=ldi,mapCenterLatitude=50.
↪82502,mapCenterLongitude=4.46045,initialZoom=7,jsonUrl=/public/data/json/ldi-stations.
↪json,autoPanLabels=false \
    --fields=pm2-5=P2,pm10=P1 \
    | http --session=grafana POST $GRAFANA_URL/api/dashboards/db
```

IRCELINE network

```
# Define Grafana instance and login.
export GRAFANA_URL=http://localhost:3000
http --session=grafana $GRAFANA_URL --auth=admin:admin
```

Create datasource:

```
# Create data source object for "vmm @ InfluxDB".
luftdatenpumpe grafana --kind=datasource --name=influxdb \
    --variables=tsdbDatasource=vmm \
    | http --session=grafana POST $GRAFANA_URL/api/datasources
```

Create dashboard with graph panel:

```
luftdatenpumpe grafana --kind=dashboard --name=trend \
    --variables=tsdbDatasource=vmm,sensorNetwork=irceline,timeRefresh=1h \
    --fields=pm1=particulate-matter-1-m,pm2-5=particulate-matter-2-5-m,pm10=particulate-
↪matter-10-m \
    | http --session=grafana POST $GRAFANA_URL/api/dashboards/db
```

Export station metadata from RDBMS database (PostgreSQL) to JSON file for Panodata Map Panel:

```
stationsfile=/var/lib/grafana/data/json/vmm-stations.json
# macOS: stationsfile=/usr/local/var/lib/grafana/data/json/vmm-stations.json

luftdatenpumpe stations --source=postgresql://luftdatenpumpe@localhost/weatherbase --
↪target=json.flex+stream://sys.stdout --target-fieldmap='key=station_id|str,name=sos_
↪feature_and_id' > $stationsfile
```

Create dashboard with map and table panels for IRCELINE:

```
luftdatenpumpe grafana --kind=dashboard --name=map \
    --variables=tsdbDatasource=vmm,sensorNetwork=irceline,timeFromOffset=120m,
↪timeRefresh=1h,mapCenterLatitude=50.82502,mapCenterLongitude=4.46045,initialZoom=7,
↪jsonUrl=/public/data/json/vmm-stations.json,autoPanLabels=false \
    --fields=pm1=particulate-matter-1-m,pm2-5=particulate-matter-2-5-m,pm10=particulate-
↪matter-10-m \
    | http --session=grafana POST $GRAFANA_URL/api/dashboards/db
```

2.2.3 Notes

Notes about PostgreSQL authentication

Introduction

When configuring and operating Luftdatenpumpe on a production machine, you may want to look into configuring Trust Authentication for PostgreSQL.

> When `trust` authentication is specified, PostgreSQL assumes that anyone who can connect to the server is authorized to access the database with whatever database user name they specify (even superuser names).

Of course, restrictions made in the `database` and `user` columns still apply. This method should only be used when there is adequate operating-system-level protection on connections to the server.

`trust` authentication is appropriate and convenient for local connections on a single-user workstation.

Configuration

To configure `trust` authentication for the users `luftdatenpumpe` and `grafana`, please add those lines to your `pg_hba.conf`:

```
host    weatherbase       luftdatenpumpe  127.0.0.1/32      trust
host    weatherbase       luftdatenpumpe  ::1/128           trust
local   weatherbase       luftdatenpumpe                    trust

host    weatherbase       grafana         127.0.0.1/32      trust
host    weatherbase       grafana         ::1/128           trust
local   weatherbase       grafana                           trust
```

InfluxDB notes

High-performance InfluxDB

For high performance ingestion into InfluxDB, its UDP data sink is your friend.

Usage

```
# Acquire data from live API and store into InfluxDB, with UDP
luftdatenpumpe readings --target=udp+influxdb://localhost:4445/luftdaten_info --progress
```

Configuration

Configure UDP data sink with InfluxDB in `influxdb.conf`:

```
# High-traffic data feed for ingesting data from luftdaten.info
# https://docs.influxdata.com/influxdb/v1.7/supported_protocols/udp/
[[udp]]

  # UDP FTW
  enabled = true

  # UDP port we are listening to
  bind-address = ":4445"

  # Name of the database that will be written to
  database = "luftdaten_info"

  # Will flush after buffering this many points
  batch-size = 5000
```

(continues on next page)

(continued from previous page)

```
  # Number of batches that may be pending in memory
  batch-pending = 100

  # Will flush each N seconds if batch-size is not reached
  batch-timeout = "15s"

  # UDP read buffer size: 8 MB (8*1024*1024)
  #read-buffer = 8388608
  read-buffer = 0
```

Redis notes

Introduction

This program extensively uses a runtime cache based on Redis.

Data durability

To make this work best, you should enable data durability with your Redis instance.

> The append-only file is an alternative, fully durable strategy for Redis. It became available in version 1.1. You can turn on the AOF in your Redis configuration file (e.g. `/etc/redis/redis.conf`):

```
appendonly yes
```

Running

In order to run Redis from your local working tree, you might want to invoke:

```
echo 'dir ./var/lib\nappendonly yes' | redis-server -
```

Looking glass

In order to look into what is going on at the Redis substrate, you might want to invoke:

```
redis-cli monitor
```

Please take care, the output is noisy.

Running in production

We experienced infrequent crashes of our Redis instance on CentOS Linux 7.6.1810. In order to work around that problem, we configured systemd to restart the Redis instance on failure by adding a file to the `/etc` directory as outlined below.

`cat /etc/systemd/system/redis.service.d/restart.conf`:

```
[Service]
# https://jonarcher.info/2015/08/ensure-systemd-services-restart-on-failure/
# Please run "systemctl daemon-reload" after making changes to this file.
Restart=always
RestartSec=3
```

Please run `systemctl daemon-reload` after adding this file or making changes to it.

We have been tracking this issue at [1].

[1] https://github.com/earthobservations/luftdatenpumpe/issues/7

Python virtualenv

About

virtualenv is a tool to create isolated Python environments. We recommend it for installing the software and its dependencies independently of your Python distribution.

Install

Create Python3 virtualenv:

```
python3 -m venv .venv
```

Install:

```
# Activate virtualenv
source .venv/bin/activate

# Install Python package
pip install $program
```

Note: Don't forget to activate the virtualenv again when trying to use the program. Alternatively, use the full path to */path/to/.venv/bin/$program*.

2.3 Usage

```
$ luftdatenpumpe --help

Usage:
  luftdatenpumpe networks [--network=<network>]
  luftdatenpumpe stations --network=<network> [options] [--target=<target>]...
  luftdatenpumpe readings --network=<network> [options] [--target=<target>]... [--
↪timespan=<timespan>]
  luftdatenpumpe database --network=<network> [--target=<target>]... [--create-views] [--
↪grant-user=<username>] [--drop-data] [--drop-tables] [--drop-database]
  luftdatenpumpe grafana --network=<network> --kind=<kind> --name=<name> [--variables=
↪<variables>] [--fields=<fields>]
  luftdatenpumpe --version
  luftdatenpumpe (-h | --help)

Options:
  --network=<network>           Which sensor network/database to use.
                                Inquire available networks by running "luftdatenpumpe␣
↪networks".
  --source=<source>             Data source, either "api" or "file://" [default: api].
  --country=<countries>         Filter data by given country codes, comma-separated.
  --station=<stations>          Filter data by given location ids, comma-separated.
  --sensor=<sensors>            Filter data by given sensor ids, comma-separated.
  --sensor-type=<sensor-types>  Filter data by given sensor types, comma-separated.
  --timespan=<timespan>         Filter readings by time range, only for SOS API (e.g.␣
↪IRCELINE).
  --reverse-geocode             Compute geographic address using the Nominatim reverse␣
↪geocoder
  --target=<target>             Data output target
  --target-fieldmap=<fieldmap>  Field name mapping for "json+flex" target
  --disable-nominatim-cache     Disable Nominatim reverse geocoder cache
  --progress                    Show progress bar
  --version                     Show version information
  --dry-run                     Skip publishing to MQTT bus
  --debug                       Enable debug messages
  -h --help                     Show this screen


Network list:

  # Display list of supported sensor networks
  luftdatenpumpe networks

Acquire stations (LDI):

  # Display metadata for given countries in JSON format
  luftdatenpumpe stations --network=ldi --country=BE,NL,LU

  # Display metadata for given stations in JSON format, with reverse geocoding
  luftdatenpumpe stations --network=ldi --station=49,1033 --reverse-geocode
```

(continues on next page)

(continued from previous page)

```
Acquire readings (LDI):

  # Display measurement readings for specific station identifiers.
  luftdatenpumpe readings --network=ldi --station=49,1033 --reverse-geocode

  # Display measurement readings for specific sensor identifiers.
  luftdatenpumpe readings --network=ldi --sensor=417

Acquire stations and readings (IRCELINE):

  luftdatenpumpe stations --network=irceline
  luftdatenpumpe readings --network=irceline --station=1030,1751 --reverse-geocode

Acquire stations and readings (OpenAQ):

  luftdatenpumpe stations --network=openaq
  luftdatenpumpe readings --network=openaq --country=IN,PK

Heads up:

  From now on, let's pretend we always want to operate on data coming from the
  sensor network "luftdaten.info", which is identified by "--network=ldi". To
  make this more convenient, we use an environment variable to signal this
  to subsequent invocations of "luftdatenpumpe" by running::

    export LDP_NETWORK=ldi

Getting started:

  # Display metadata for given stations in JSON format, with reverse geocoding
  luftdatenpumpe stations --station=49,1033 --reverse-geocode --target=json+stream://sys.
↪stderr


Convert stations into format suitable for Grafana:

  # Display list of stations in JSON format made of value/text items, suitable for use␣
↪as a Grafana JSON data source
  luftdatenpumpe stations --station=49,1033 --reverse-geocode --target=json.grafana.
↪vt+stream://sys.stdout

  # Display list of stations in JSON format made of key/name items, suitable for use as␣
↪a mapping in Panodata Map Panel
  luftdatenpumpe stations --station=49,1033 --reverse-geocode --target=json.grafana.
↪kn+stream://sys.stdout


Write stations into / read stations from RDBMS database:

  # Store list of stations and metadata into RDBMS database (PostgreSQL)
  luftdatenpumpe stations --station=49,1033 --reverse-geocode --target=postgresql://
↪luftdatenpumpe@localhost/weatherbase
```

(continues on next page)

(continued from previous page)

```
  # Read station information from RDBMS database (PostgreSQL) and format for Panodata␣
↪Map Panel
  luftdatenpumpe stations --source=postgresql://luftdatenpumpe@localhost/weatherbase --
↪target=json.grafana.kn+stream://sys.stdout


Live data examples (InfluxDB):

  # Store into InfluxDB running on "localhost"
  luftdatenpumpe readings --station=49,1033 --target=influxdb://localhost/luftdaten_info

  # Store into InfluxDB, with UDP
  luftdatenpumpe readings --station=49,1033 --target=udp+influxdb://localhost:4445/
↪luftdaten_info

  # Store into InfluxDB, with authentication
  luftdatenpumpe readings --station=49,1033 --target=influxdb://luftdatenpumpe@localhost/
↪luftdaten_info


LDI CSV archive data examples (InfluxDB):

  # Mirror archive of luftdaten.info, limiting to 2015 only
  wget --mirror --continue --no-host-directories --directory-prefix=/var/spool/archive.
↪luftdaten.info --accept-regex='2015' http://archive.luftdaten.info/

  # Ingest station information from CSV archive files, store into PostgreSQL
  luftdatenpumpe stations --network=ldi --source=file:///var/spool/archive.luftdaten.
↪info --target=postgresql://luftdatenpumpe@localhost/weatherbase --reverse-geocode --
↪progress

  # Ingest readings from CSV archive files, store into InfluxDB
  luftdatenpumpe readings --network=ldi --source=file:///var/spool/archive.luftdaten.
↪info --target=influxdb://luftdatenpumpe@localhost/luftdaten_info --progress

  # Ingest most early readings
  luftdatenpumpe readings --network=ldi --source=file:///var/spool/archive.luftdaten.
↪info/2015-10-*

  # Ingest most early PMS sensors
  luftdatenpumpe readings --network=ldi --source=file:///var/spool/archive.luftdaten.
↪info/2017-1*/*pms*.csv


Live data examples (MQTT):

  # Publish data to topic "luftdaten.info" at MQTT broker running on "localhost"
  luftdatenpumpe readings --station=49,1033 --target=mqtt://localhost/luftdaten.info

  # MQTT publishing, with authentication
  luftdatenpumpe readings --station=49,1033 --target=mqtt://username:password@localhost/
```

(continues on next page)

(continued from previous page)

```
↪luftdaten.info


Combined examples:

  # Write stations to STDERR and PostgreSQL
  luftdatenpumpe stations --station=49,1033 --target=json+stream://sys.stderr --
↪target=postgresql://luftdatenpumpe@localhost/weatherbase

  # Write readings to STDERR, MQTT, and InfluxDB
  luftdatenpumpe readings --station=49,1033 --target=json+stream://sys.stderr --
↪target=mqtt://localhost/luftdaten.info --target=influxdb://luftdatenpumpe@localhost/
↪luftdaten_info
```

2.4 Gallery

Some example installations, usually running live data feeds through them.

2.4.1 luftdaten.info

About

Acquisition of sensor data from the LDI network.

Adapter

Implemented by luftdatenpumpe.source.luftdaten_info

Dashboards

- LDI Feinstaub Karte Europa
- LDI Feinstaub Verlauf
- Vergleich LDI Umweltdaten vs. DWD Wetterdaten
- Vergleich LDI Feuchtigkeitswerte BME280 vs. DHT22 mit DWD-Daten
- LDI Karte und Kompensation
- LDI Feuchtekorrektur PM10

Screenshots

Fig. 1: Coverage of LDI network 2015-2018

Fig. 2: Fine dust pollution on New Year's Eve 2018 | Animation of fine dust pollution on New Year's Eve 2018 across Europe

References

- https://web.archive.org/web/20220604103954/https://luftdaten.info/
- https://sensor.community/
- https://community.hiveeyes.org/t/erneuerung-der-luftdatenpumpe/1199
- https://community.hiveeyes.org/t/ldi-data-plane-v2/1412

Labs

- LDI Demo #1 » Stations by name, country and state
- LDI Demo #2 » Cascaded stations
- LDI Demo #3 » Measurements by cascaded location selector
- LDI Demo #4 » Find stations by sensor type
- LDI Demo #5 » Map by location and sensor type

2.4.2 DHT22

About

An analysis of the results of DHT22 humidity sensors.

Screenshots

2.4.3 IRCELINE

About

Air quality monitoring for the Flanders Environment Agency in Belgium.

Luftdatenpumpe helped VMM to support their work for the Belgian Interregional Environment Agency (IRCEL - CELINE) on air quality monitoring within the EU-funded Corona EU and VAQUUMS projects.

More details about this can be found at Supporting the Flanders Environment Agency (VMM) by analyzing and visualizing air quality sensor data with InfluxDB and Grafana.

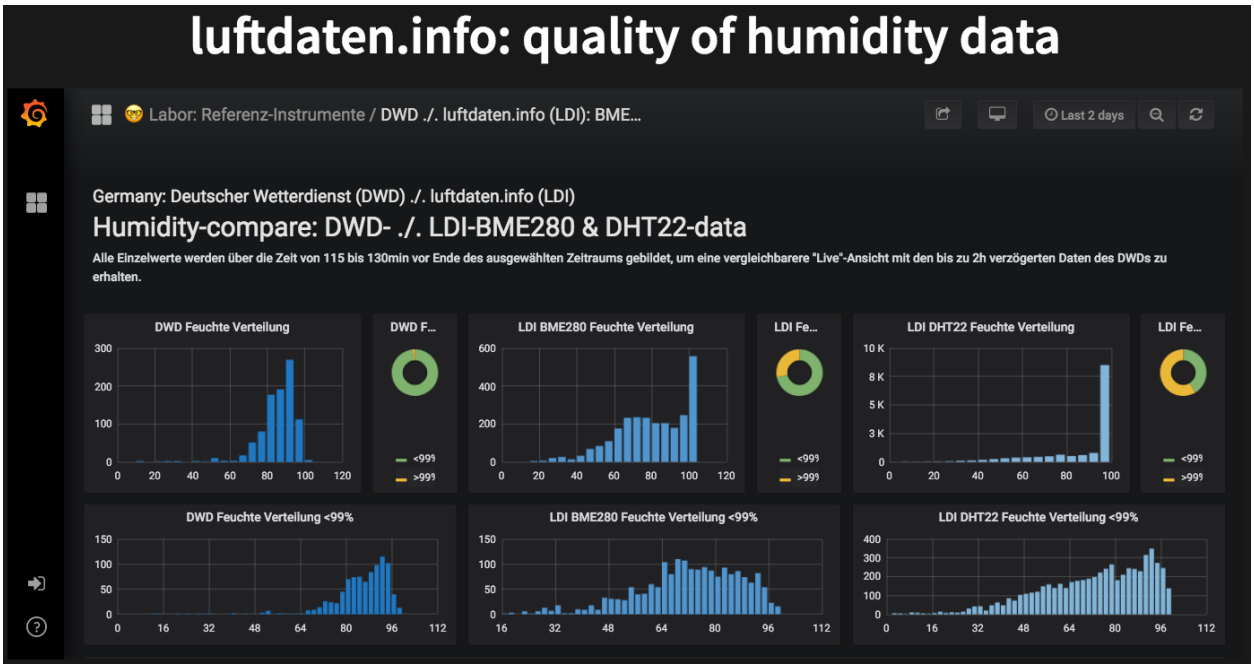


Fig. 3: Analyzing DHT22 measurement values on the LDI network.

Adapter

Implemented by luftdatenpumpe.source.irceline

Dashboards

- Luftdaten-Viewer » IRCELINE » Map
- Luftdaten-Viewer » IRCELINE » Trend

Screenshots

References

- https://www.irceline.be/en
- https://web.archive.org/web/20211129062716/http://deus.irceline.be/
- https://en.vmm.be/
- https://web.archive.org/web/20210112053937/https://project-corona.eu/default.aspx
- https://geo.irceline.be/sos/static/doc/api-doc/

Fig. 4: Comparing air quality sensors.

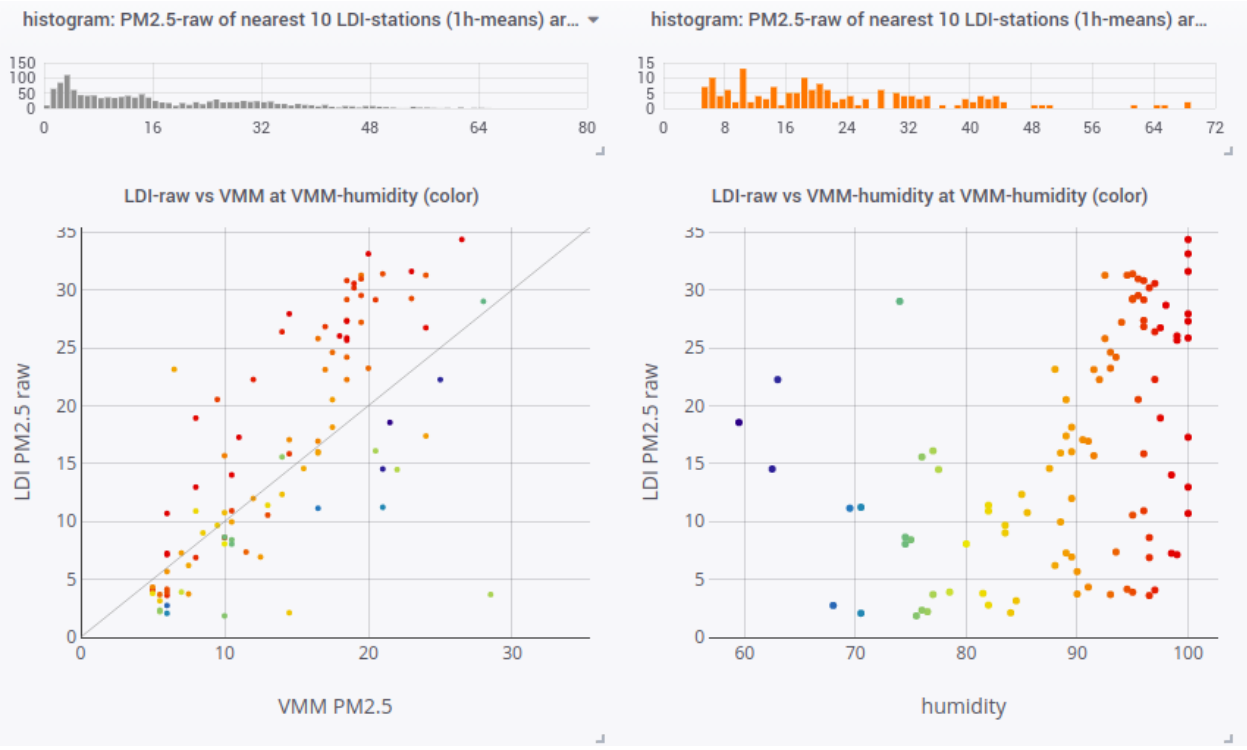


Fig. 5: Comparing air quality sensors.

Fig. 6: Humidity compensation for air quality sensors.

2.5 PostGIS/OSM tutorial

2.5.1 Introduction

Luftdatenpumpe uses a PostGIS `POINT` to store the geolocation of LDI stations.

In the query expression examples displayed below, we use the geocoordinates `POINT(9.1800132 48.7784485)` for Stuttgart, Germany.

This coordinate has been returned by querying Nominatim for city==stuttgart.

2.5.2 Details

See also:

- http://postgis.net/workshops/postgis-intro/geography.html
- http://postgis.net/workshops/postgis-intro/knn.html

Excerpts from the documentation

4.2.2. When to use Geography Data type over Geometry data type If your data is global or covers a continental region, you may find that GEOGRAPHY allows you to build a system without having to worry about projection details. You store your data in longitude/latitude, and use the functions that have been defined on GEOGRAPHY.

– https://postgis.net/docs/using_postgis_dbmanagement.html#PostGIS_GeographyVSGeometry

4.2.3.1. Do you calculate on the sphere or the spheroid? By default, all distance and area calculations are done on the spheroid.

– https://postgis.net/docs/using_postgis_dbmanagement.html#idm1644

2.5.3 PostGIS query expressions

Sort by distance

```
-- Find specified number of nearest stations through sorting by distance,
-- display name and textual representation of coordinates.
SELECT name, ST_AsText(geopoint) AS geopoint
FROM ldi_stations
ORDER BY geopoint <-> 'POINT(9.18001 48.77844)'
LIMIT 5;
```

```
-- Same as above, but display name and GeoJson representation of coordinates.
SELECT name, ST_AsGeoJson(geopoint) AS geojson
FROM ldi_stations
ORDER BY geopoint <-> 'POINT(9.18001 48.77844)'
LIMIT 5;
```

Match within range

```
-- Find all stations within specified range in meters while sorting by distance.
-- The computation uses a bounding box expanded from the specified geographic point,
-- by specifying a single distance with which the box will be expanded in all directions.
-- https://postgis.net/docs/ST_DWithin.html
-- https://postgis.net/docs/ST_Expand.html
SELECT name, ST_AsText(geopoint) AS geopoint
FROM ldi_stations
WHERE ST_DWithin(geopoint, 'POINT(9.18001 48.77844)', 3000)
ORDER BY geopoint <-> 'POINT(9.18001 48.77844)';
```

In order to save on repeating this `POINT` coordinates here, we can alias it into a virtual `position` field:

```
-- List all stations within 3000 meters around specified coordinates.
-- This is what OpenStreetMap/Nominatim thinks the center of Stuttgart is.
WITH stuttgart AS (
    SELECT ST_GeographyFromText('POINT(9.18001 48.77844)') AS position
)
SELECT name, ST_AsText(geopoint) AS geopoint
FROM ldi_stations, stuttgart
WHERE ST_DWithin(geopoint, stuttgart.position, 3000)
ORDER BY geopoint <-> stuttgart.position;
```

Mixing in OSM/Nominatim

As these expressions use the constraint `osm_city = 'Stuttgart'`, OSM data is involved. Centroid of OSM input coordinates

```
-- Compute center of city as the arithmetic mean of the input coordinates.
-- http://postgis.net/workshops/postgis-intro/geometry_returning.html
-- http://postgis.net/workshops/postgis-intro/advanced_geometry_construction.html
-- https://postgis.net/docs/ST_Collect.html
```

(continues on next page)

(continued from previous page)

```
-- https://postgis.net/docs/ST_Centroid.html
SELECT ST_AsText(ST_Centroid(ST_Collect(geopoint::geometry))::geography) AS geopoint
FROM ldi_network
WHERE osm_city = 'Stuttgart';
```

```
-- Using the formula above, find specified number of nearest stations by city name.
-- https://stackoverflow.com/a/19859047
-- https://postgis.net/docs/ST_X.html
-- https://postgis.net/docs/ST_Y.html
WITH center_of AS (
    SELECT ST_Centroid(ST_Collect(geopoint::geometry))::geography AS position FROM ldi_
↪network WHERE osm_city = 'Stuttgart'
)
SELECT name, ST_AsText(geopoint) AS geopoint
FROM ldi_stations, center_of
ORDER BY geopoint <-> center_of.position
LIMIT 5;
```

```
-- List all stations within 3000 meters around what LDI thinks the center of Stuttgart␣
↪is.
WITH stuttgart AS (
    SELECT ST_Centroid(ST_Collect(geopoint::geometry))::geography AS position FROM ldi_
↪network WHERE osm_city = 'Stuttgart'
)
SELECT name, ST_AsText(geopoint) AS geopoint
FROM ldi_stations, stuttgart
WHERE ST_DWithin(geopoint, stuttgart.position, 3000)
ORDER BY geopoint <-> stuttgart.position;
```

2.5.4 Accessing the OSM/Nominatim API

By using the PostgreSQL extension pgsql-http, which is effectively a »HTTP client for PostgreSQL«, you can directly access the Nominatim HTTP API for asking for a `geotext` field from a specified city or other location by using the `polygon_text=1` query parameter.

The `geotext` field yielded by the response of the API is in PostGIS-compatible `POINT(lon lat)` format already.

Print coordinate by asking for `city=stuttgart`, effectively roundtripping through HTTP and PostGIS:

```
-- https://github.com/pramsey/pgsql-http
-- https://wiki.openstreetmap.org/wiki/Nominatim
-- https://www.postgresql.org/docs/9.3/functions-json.html
-- TODO: Provide this as a native Grafana datasource and/or variable somehow?
CREATE EXTENSION http;

-- Nominatim request subselect.
WITH stuttgart AS (
    SELECT
        ST_GeographyFromText(content::json->0->>'geotext') AS position
    FROM
        http_get('https://nominatim.hiveeyes.org/search.php?format=jsonv2&
```

(continues on next page)

(continued from previous page)

```
↪addressdetails=1&polygon_text=1&city=stuttgart')
)

-- Print position as text.
-- You should do more sophisticated things here, see below.
SELECT ST_AsText(position) FROM stuttgart;
```

Match within range

```
-- List all stations within 3000 meters around specified city.
-- The coordinates of the city is coming from OpenStreetMap/Nominatim.
WITH stuttgart AS (
    SELECT ST_GeographyFromText(content::json->0->>'geotext') AS position
    FROM http_get('https://nominatim.hiveeyes.org/search.php?format=jsonv2&
↪addressdetails=1&polygon_text=1&city=stuttgart')
)
SELECT name, ST_AsText(geopoint) AS geopoint
FROM ldi_stations, stuttgart
WHERE ST_DWithin(geopoint, stuttgart.position, 3000)
ORDER BY geopoint <-> stuttgart.position;
```

2.5.5 Using the local database with OSM data

Using the `--reverse-geocode` option, Luftdatenpumpe will query Nominatim on its own behalf, and will populate the PostGIS database with information from OSM.

This data can be queried conveniently without having to reach out to Nominatim using pgsql-http.

Todo: Show some example queries.

2.6 MQTT publishing

Luftdatenpumpe can publish processed records to an MQTT topic.

2.6.1 Usage

Subscribe:

```
mosquitto_sub -h mqtt.example.org -t '#' -v
```

Publish:

```
luftdatenpumpe readings --network=ldi --station="49,1033" \
    --target=mqtt://mqtt.example.org/luftdaten.info
```

2.6.2 Example

Results (reformatted for better readability):

```
luftdaten.info {"latitude": 48.53, "longitude": 9.2, "altitude": 373.1, "country": "DE",
↪"exact_location": 0, "indoor": 0, "geohash": "u0ws16e5xx9n", "location_id": 49, "time
↪": "2022-07-10T14:35:46Z", "sensor_id": 107, "sensor_type": "PPD42NS", "durP1": 100367.
↪0, "ratioP1": 0.33, "P1": 174.21, "durP2": 0.0, "ratioP2": 0.0, "P2": 0.62}
luftdaten.info {"latitude": 48.53, "longitude": 9.2, "altitude": 373.1, "country": "DE",
↪"exact_location": 0, "indoor": 0, "geohash": "u0ws16e5xx9n", "location_id": 49, "time
↪": "2022-07-10T14:35:47Z", "sensor_id": 108, "sensor_type": "DHT22", "temperature": 24.
↪1, "humidity": 1.0}
luftdaten.info {"latitude": 48.53, "longitude": 9.2, "altitude": 373.1, "country": "DE",
↪"exact_location": 0, "indoor": 0, "geohash": "u0ws16e5xx9n", "location_id": 49, "time
↪": "2022-07-10T14:36:02Z", "sensor_id": 417, "sensor_type": "SDS011", "P1": 7.8, "P2":␣
↪2.53}
luftdaten.info {"latitude": 48.53, "longitude": 9.2, "altitude": 373.1, "country": "DE",
↪"exact_location": 0, "indoor": 0, "geohash": "u0ws16e5xx9n", "location_id": 49, "time
↪": "2022-07-10T14:36:03Z", "sensor_id": 418, "sensor_type": "DHT22", "temperature": 25.
↪0, "humidity": 20.3}

luftdaten.info {"latitude": 52.56, "longitude": 13.374, "altitude": 44.3, "country": "DE
↪", "exact_location": 0, "indoor": 0, "geohash": "u33e0268hy1h", "location_id": 1033,
↪"time": "2022-07-10T14:36:04Z", "sensor_id": 2055, "sensor_type": "SDS011", "P1": 3.4,
↪"P2": 1.4}
luftdaten.info {"latitude": 52.56, "longitude": 13.374, "altitude": 44.3, "country": "DE
↪", "exact_location": 0, "indoor": 0, "geohash": "u33e0268hy1h", "location_id": 1033,
↪"time": "2022-07-10T14:36:05Z", "sensor_id": 2056, "sensor_type": "DHT22", "temperature
↪": 20.2, "humidity": 3.9}
luftdaten.info {"latitude": 52.56, "longitude": 13.374, "altitude": 44.3, "country": "DE
↪", "exact_location": 0, "indoor": 0, "geohash": "u33e0268hy1h", "location_id": 1033,
↪"time": "2022-07-10T14:36:05Z", "sensor_id": 67015, "sensor_type": "BME280",
↪"temperature": 20.39, "pressure": 101364.47, "humidity": 41.09, "pressure_at_sealevel
↪": 101888.09}
```

CHAPTER

THREE

DEVELOPMENT

3.1 Contributors

Here is an alphabetically ordered list of the people who contributed to this software in one way or another.

- Andreas Motl <andreas.motl@panodata.org>
- David Roet <https://en.vmm.be/>
- Matthias Mehldau <matthias.mehldau@panodata.org>
- Matthias Steffen <https://github.com/panzki>
- Maurits Descamps <https://github.com/MauritsDescamps>
- Olav Peeters <https://en.vmm.be/>
- Oliver <https://github.com/ohobby>
- Richard Pobering <richard.pobering@panodata.org>

Thanks a stack.

3.1.1 Acknowledgements

Creating this program would not have been possible without the help of the amazing people listed above.

Many more people have been involved creating the foundation infrastructure and the software components this program is leveraging, it is built upon Grafana, InfluxDB, PostgreSQL, PostGIS, Python, and all the associated modules around them.

Standing on the shoulders of giants. Thank you so much, you know who you are.

3.2 Changelog

3.2.1 in progress

- Fix "Luftdaten-Viewer Grafana" documentation section about exporting station metadata from PostGIS to JSON file. Thanks, @ohobby.
- Add two new OSM-based synthetic geolocation fields for better addressing of micro-regions, `district_postcode_city_sensorid`, and `suburb_postcode_city_sensorid`. Thanks, @ohobby.
- Add rendered documentation at https://luftdatenpumpe.readthedocs.io/

3.2.2 2022-12-05 0.21.1

- Fix acquisition of station list from EEA

3.2.3 2022-08-03 0.21.0

- Fix Python sandbox and test infrastructure
- Improve Nominatim reverse geocoding heuristics and corresponding tests
- Fix dependencies. Thanks, @MauritsDescamps.
- LDI: Fix data acquisition
- LDI: Fix forwarding to MQTT
- OpenAQ: Fix acquisition for certain records without geo information
- Improve project tooling, add CI, linter, and code formatter
- Sanitize code and prose with linters

3.2.4 2020-01-28 0.20.2

- Don't use caching for live data. Thanks, Matthias.

3.2.5 2020-01-08 0.20.1

- Add missing `openaq.py` file

3.2.6 2020-01-08 0.20.0

- Add adapter for OpenAQ
- Slightly improve reverse geocoding
- Use improved "Panodata Map Panel" instead of the original "Grafana Worldmap Panel"
- Update documentation

3.2.7 2019-12-09 0.19.1

- Fix invalid JSON file export for stations list (#17). Thanks, David.

3.2.8 2019-10-30 0.19.0

- Improve EEA station processing
- Update "pflock" command to report if invocation failed
- Use HTTP POST when creating the InfluxDB datasource. Thanks, David.

3.2.9 2019-09-30 0.18.2

- Improve memory consumption
- Don't process non-float values

3.2.10 2019-09-29 0.18.1

- Mask observation values with magic number `-99.99` representing `NaN` values from SOS/IRCELINE, #3.

3.2.11 2019-09-29 0.18.0

- Update documentation for installing Grafana Worldmap Plugin
- Add Redis setup documentation re. #7. Thanks, David.
- Add PostgreSQL setup documentation re. #8. Thanks, David.
- Fix LDI historical data import, #10. Thanks, Wetter.

3.2.12 2019-09-27 0.17.0

- Update `GRAFANA_URL` in documentation
- Improve data wrangling robustness for luftdaten.info
- Acquire and process station list from EEA

3.2.13 2019-06-27 0.16.0

- Nothing changed

3.2.14 2019-06-27 0.15.0

- Handle incomplete records gracefully when writing to InfluxDB
- Improve egress JSON field conversion

3.2.15 2019-05-21 0.14.1

- Fix erroneous dependency package version bump

3.2.16 2019-05-21 0.14.0

- Fix silly mixup with "is_active" indicator
- Ignore `--country=BE` when operating on IRCELINE
- Add new synthetic database-view fields "road_and_name_and_id" and "sos_feature_and_id"
- Add new `json.flex` output target for flexible fieldname mapping

3.2.17 2019-05-20 0.13.0

- Improve IRCELINE request handling robustness
- Add "sensor_first_date" and "sensor_last_date" fields for IRCELINE to indicate <= 7 days of data freshness by synthesized field "is_active".

3.2.18 2019-05-19 0.12.1

- Improve IRCELINE ingest with `--timespan` option vs. `batch_size`

3.2.19 2019-05-19 0.12.0

- Always fetch last 12 hours worth of data to reduce gaps when API is offline.
- Update documentation
- Distinguish between "sensor_type_name" and "sensor_type_id"
- Tune map panel default settings

3.2.20 2019-04-26 0.11.0

- Push architecture towards ingesting of data from multiple sensor networks
- Integrate data from the SOS REST API of the IRCELINE network

3.2.21 2019-04-22 0.10.0

- Improve RDBMS subsystem
- Improve robustness, logging and error handling
- Add resources and documentation for running as cron job
- Allow customizing the Grafana panels from the command line

3.2.22 2019-04-10 0.9.0

- Add GIS capabilities through PostGIS
- Set default format for "stream://" targets to "json"
- Fix published messages getting lost when not starting the MQTT main loop after connecting to MQTT broker
- Refactor station list filter
- Filter stations by country code

3.2.23 2019-01-22 0.8.2

- Add missing sensor DS18B20
- Fix PostgreSQL version in Grafana datasource JSON
- Add station id to "multiple stations" chooser on trend dashboard
- Don't try to enrich incomplete station information

3.2.24 2019-01-19 0.8.1

- Make dashboards not editable

3.2.25 2019-01-19 0.8.0

- Refactor and improve Grafana datasource- and dashboard JSON files
- Add `luftdatenpumpe grafana` subcommand for accessing Grafana datasource- and dashboard JSON files
- Improve documentation significantly

3.2.26 2019-01-18 0.7.0

- Rename OSM data field "country_name" back to "country"
- Add sanity checks for protecting against unqualified responses from Nominatim service with DE-only dataset loaded
- Use country code for routing to different Nominatim services, one of them having the DE-only dataset loaded
- Improve RDBMS database schema
- Naming things
- Show cardinality in sensor type chooser

3.2.27 2019-01-18 0.6.0

- Fix renaming OSM field “country” to “country_name”

3.2.28 2019-01-18 0.5.0

- Add InfluxDB egress handler
- Improve HTTP response caching
- Probe Redis before starting and croak if connection fails
- Add “geohash” field when writing into InfluxDB
- Use nominatim.hiveeyes.org as primary reverse geocoder, fall back to nominatim.openstreetmap.org
- Add option to disable the Nominatim cache
- Add configuration and documentation about Grafana Worldmap
- Unlock CSV data acquisition from archive.luftdaten.info
- Add Grafana Graph dashboard
- Add User-Agent for requests to api.luftdaten.info
- Improve globbing when selecting path for CSV import
- Compensate empty values (nan) when importing from CSV
- Add output formatter for Grafana Worldmap Panel JSON file
- Add RDBMS database (PostgreSQL) as station data source
- Add `--sensor-type` filter option
- Improve CSV file reading
- Flush each 50 records when talking to InfluxDB with UDP
- Introduce quick mode for importing just the first few records
- Add new option “–create-database-view”
- Rename OSM data field “country” to “country_name”

3.2.29 2018-12-11 0.4.3

- Fix setup.py
- Add MANIFEST.in file

3.2.30 2018-12-11 0.4.2

- Use “geohash2” package from PyPI for Python3 compatibility
- Fix twine. Just works outside of virtualenv.

3.2.31 2018-12-11 0.4.1

- Remove unknown Trove classifiers from setup.py

3.2.32 2018-12-11 0.4.0

- Refactoring, Python2/3 compatibility, Add setup.py
- Add “sensor_type” information to station list
- Use Redis-based caching through dogpile.cache, ditch Beaker
- Refactor data munging
- Always cache full response from Nominatim
- Cache responses from the luftdaten.info API for five minutes
- Add basic RDBMS adapter for storing station list and associated information to Postgres and other SQL databases supported by SQLAlchemy
- Streamline station data schema
- Add test harness for reverse geocoder subsystem
- Improve robustness and quality of reverse geocoder
- Make “sensors” data substructure an array
- Refactor target machinery and redesign command line interface
- Add release tooling

3.2.33 2018-12-02 0.3.0

- Add option “–dry-run”
- Fix filtering by station id
- Fix access to Nominatim reverse geocoder API
- Use “appdirs” module for computing cache location. Report about cache location at startup.
- Improve OSM address formatter: Honor “footway” as another fieldname choice for encoding the “road”
- Improve OSM address formatter: Honor “suburb” field
- Improve filtering by sensor- and/or location-identifiers
- Implement “stations” subcommand to acquire, display and export list of stations
- Prevent duplicate segments in formatted address
- Use station id as label when name is not available

3.2.34 2017-06-06 0.2.0

- Add filtering by sensor id. Thanks, Panzki.

3.2.35 2017-04-25 0.1.0

- Add commandline interface
- Add caching for Nominatim responses
- Appropriate timestamp mungling
- Improve Documentation

3.2.36 2017-03-31 0.0.0

- Basic implementation to request data from live API of luftdaten.info, enrich geospatial information and publish to MQTT bus
- Add "sensor_type" field
- Improve OSM address formatter

3.3 Backlog

3.3.1 Iteration +0

- [x] CI: Fix anomalies with Nominatim geocoder
- [x] Add CI/GHA and badges
- [x] Code formatting, satisfy linter
- [x] Render `luftdatenpumpe --help` to documentation
- [x] Release 0.21.0
- [x] Add rendered documentation at https://luftdatenpumpe.readthedocs.io/
- [o] Problem with sensor ID reference in new synthesized geolocation fields https://github.com/earthobservations/luftdatenpumpe/issues/57
- [o] Release 0.22.0
- [o] Support for InfluxDB 2.x
- [o] Support for Grafana 9.x
- [o] Support for EEA
  - Stations list: Compute `AirPollutant` from `AirPollutantCode`
- [o] Add more tests
- [o] Announcements
  - https://forum.sensor.community/
  - https://community.panodata.org/t/luftdatenpumpe/21
- [o] Sandbox: Use Docker Compose for starting auxiliary services

- [o] Shipping: Provide container images
- [o] Completely migrate from `LDI` to `SC`, including backwards-compatibility
- [o] EEA, Luchtmeetnet, UmweltBundesamt

3.3.2 Iteration +1

- [o] LDI: Naming things. Switch from "luftdaten.info" to "sensor.community".
- [o] OpenAQ: Raise `limit=10000` to address more than 10000 stations
- [o] OpenAQ: Add `--timespan` option
- [o] OpenAQ: Filter by station id
- [o] OpenAQ: Unlock both `get_latest_readings` and `get_current_measurement_readings`
- [o] EEA: Add initial implementation
- [o] BLUME: Add initial implementation
  - https://github.com/johnjohndoe/blume_sos_adapter
  - https://github.com/dirkschumacher/blume_messnet_api
  - https://github.com/dirkschumacher/blume_crawler
- [o] Olav re. IRCELINE: I removed the "is active"-check for station-data, cause current data seems stalled/too old)
- [x] Add `CONTRIBUTORS.rst` file.
- [o] Make docs from https://github.com/earthobservations/luftdatenpumpe/issues/9
- [o] Add compact attribution information to JSON output.
  - https://www.eea.europa.eu/legal/
  - https://creativecommons.org/licenses/by/2.5/dk/deed.en_GB
- [o] Add `datasource-influxdb-flux.json` et al. and document the setup process https://source.irceline.be/corona-eu/luftdatenpumpe/issues/12
- [o] Improve documentation https://source.irceline.be/corona-eu/luftdatenpumpe/issues/1

3.3.3 Iteration +2

- [o] Data plane refactoring
  - Fused/combined data model of sensors vs. data/readings
- [x] IRCELINE: Progressively/chunked fetching of timeseries information - not all at once.
- [o] Refactor Markdown documentation in Grafana Dashboards
- [o] Improve IRCELINE data processing efficiency
- [o] Error with invalid timestamps when requesting IRCELINE: "statusCode" error
- [o] Grafana: Drill down to detail view via circle https://vmm.hiveeyes.org/grafana/d/gG-dP2kWk/luftdaten-viewer-ldi-trend?var-ldi_station_id=8667
- [o] Grafana: Enhanced popover with structured (meta)data transfer
- [o] Grafana: Drill down to detail view via popover

- [o] As the Panodata Map Panel (ex. Grafana Worldmap Panel) decodes the geohash to lat/lon using `decodeGeoHash()` anyway, let's go back to storing the position as lat/lon again.
- [o] Use Grafana Folder "Luftdatenpumpe" for storing dashboards.
- [o] When acquiring data from specific sensors, use API endpoints like `http://api.luftdaten.info/v1/sensor/25735/`.
- [o] Look at world air quality data

3.3.4 Iteration +3

IRCELINE

Supporting the Flanders Environment Agency (VMM). Thanks likewise for supporting us.

http://shiny.irceline.be/examples/

- [o] Optionally call SOS API with locale=de,fr,en
- [o] Add EPSG:31370
- [o] Paging does not work on `/timeseries`, neither using `limit` nor `size`.
  - http://geo.irceline.be/sos/static/doc/api-doc/#general-paging
  - https://wiki.52north.org/SensorWeb/SensorWebClientRESTInterfaceV0#Paging
- [o] Improve caching for SOS REST API taking the measurement interval into account
- [o] Add IRCELINE RIOIFDM layers

Documentation

- [o] Adjust inline Markdown: Rename "About Luftdatenpumpe" to "Luftdaten-Viewer » About" and rephrase content appropriately.
- [o] By default, gets the last reading. Querying IRCELINE w/o timestamp yields a whole week. IRCELINE has hourly, while LDI has 5-minute measurement intervals.
- [o] Improve inline documentation on `irceline.py`
- [o] Add "read this section carefully" to documentation pages
- [o] Add Sphinx documentation renderer, publish to hiveeyes.org/doc/luftdatenpumpe
- [o] Add more "About us" to luftdaten-info-trend dashboard
- [o] Cross-reference map- and trend-dashboards
- [o] Link to https://pypi.org/project/luftdaten/ and https://github.com/dr-1/airqdata
- [o] Remark about database license from https://github.com/dr-1/airqdata
- [o] Decrease logo size: https://pypi.org/project/luftdatenpumpe/
- [o] Announce @ GitHub: `luftdatenpumpe readings --station=28 --target=mqtt://mqtt.example.org/luftdaten.info/testdrive --target=stream://sys.stdout | jq .`

3.3.5 Iteration +4

- [o] Use different tile server- and/or style?
- [o] Add PM2.5 panel again?
- [o] Add "ALL" option for "Choose multiple stations" chooser
- [o] https://github.com/dr-1/airqdata/blob/master/airqdata/luftdaten.py
  - clean_measurements
  - search_proximity
  - evaluate_near_sensors
- [o] Geospatial queries against SOS REST
  - Stations around a given point http://geo.irceline.be/sos/static/doc/api-doc/#stations
- [o] Add supervisord configuration (or Docker container) for running Redis, PostGIS, InfluxDB and Grafana
- [o] Add stored procedure "osm_city_live" using the HTTP API.
- [o] Better zoom level selector for map widgets. Autozoom by station selector?
- [o] Larger form field sizes, e.g. for "query", see https://weather.hiveeyes.org/grafana/d/EWFuSqlmz/ldi-6-gis-distance-by-threshold?editview=templating&orgId=1&panelId=&fullscreen=&edit=dlslöö
- [o] `make clear-cache`
- [o] Improve selectors: stations+sensors, observations or all together

Spatial index on a geography table

```
CREATE INDEX nyc_subway_stations_geog_gix
ON nyc_subway_stations_geog USING GIST (geog);
```

– http://postgis.net/workshops/postgis-intro/geography.html

3.3.6 Iteration +5

- [o] grafanimate: Monthly gif for fast progress and daily video for atmo.
- [o] grafanimate: Add "coverage" dashboard
- [o] grafanimate: Render 2015-2018 for each year
- [o] Stats: Until 2016, it's around 1M files, 600MB data in InfluxDB and 17M P1 readings
- [o] Is it actually ok to read each sensor equally?
- [o] Downsample data on CSV import to reduce data size?
- [o] Read Parquet files from http://archive.luftdaten.info/parquet/
- [o] Vanity URLs
  - https://deutschland.maps.luftdaten.info
  - https://china.maps.luftdaten.info
  - https://europe.maps.luftdaten.info

  - https://france.maps.luftdaten.info/

3.3.7 Iteration +6

- [o] Use https://grafana.com/grafana/plugins/ryantxu-ajax-panel/ to show other content
- [o] What to do with high P1/P2 values > 1.000 and more?
- [o] CSV import: Add more sensor types
- [o] Link from sticky overlay to station trend dashboard
- [o] Refactor for handling multiple data sources and targets
- [o] Run some metric about total count of measuremnts per feed action
- [o] Use more export formats from tablib
- [o] Output data in tabular, markdown or rst formats
- [o] Publish to MQTT with separate topics
- [o] Store "boundingbox" attribute to RDBMS database
- [o] Dry-run for RDBMS storage
- Command line filters
  - [o] by sensor type
  - [o] by time range. e.g. for CSV file import.
- Panodata Map Panel
  - [o] Handle multiple languages with Nominatim. Use English as default.
  - [o] Get English (or configurable) country labels from Nominatim
  - [o] JSON endpoint: Add formatter `jq '[ .[] | {key: .station_id | tostring, name: .name} ]'`
  - [o] JSON endpoint: Map by geohash only
  - [o] Link to Nominatim place_id, see https://nominatim.hiveeyes.org/details.php?place_id=8110875
- [o] Migration documentation from https://getkotori.org/docs/applications/luftdaten.info/
- [o] Mention other projects
  - https://luftdata.se/
- [o] How to improve Panodata Map Panel JSON document becoming stale? `/public/json/ldi-stations.json?_cache=4`
- [o] Check out wizzy for Grafana provisioning? https://github.com/utkarshcmu/wizzy
- [o] Docs? https://github.com/grafana/worldmap-panel/issues/176

Email address for Nominatim

email=<valid email address>

If you are making large numbers of request please include a valid email address or alternatively include your email address as part of the User-Agent string. This information will be kept confidential and only used to contact you in the event of a problem, see Usage Policy for more details.

https://wiki.openstreetmap.org/wiki/Nominatim

3.3.8 Iteration +7

- [o] OSM: Why are some roads or towns empty? `weatherbase=# select * from ldi_osmdata where road is null limit 7;`
- [o] Add remark after "licence": "Data u00a9 OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright" like "remark": "The address information has been modified by luftdatenpumpe 0.4.0"
- [o] OSM: English labels for e.g. Hercegovine, BA
- [o] Database view https://www.postgresql.org/docs/9.2/sql-createview.html on top of https://community.hiveeyes.org/t/erneuerung-der-luftdatenpumpe/1199/25
- [o] Integrate https://github.com/openaq/openaq-fetch somehow

3.3.9 Iteration +8

- [o] Write metadata directly to PostGIS https://dataset.readthedocs.io/en/latest/
- [o] Add support for JSON and GIS data to "dataset" module
- [o] OSM: Italia only has 3-letter state names like CAL, CAM, LOM, etc.
- [o] Add PostgREST
- [o] Grafana: Link to https://www.madavi.de/sensor/graph.php and/or - `http://deutschland.maps.luftdaten.info/#13/50.9350/13.3913` and/or - https://maps.luftdaten.info/grafana/d-solo/000000004/single-sensor-view?orgId=1&panelId=1&var-node=18267 somehow?
- [o] After importing historical data, make a video from the expanding map
- [o] Update
  - https://github.com/opendata-stuttgart/sensors-software/issues/33
  - https://twitter.com/SchindlerTimo/status/1064634624192774150
- [o] Provide jq examples

Grafana

```
Appendix
========

Add text widget containing total number of stations in database.

Variable ``station_count```::
```

(continues on next page)

(continued from previous page)

```
SHOW TAG VALUES CARDINALITY WITH KEY = station_id;
```

3.3.10 Documentation

- [x] poe docs-html
- [x] poe docs-linkcheck: `cd docs; sphinx-build -b linkcheck . _build`
- [x] Update links in README.rst
- [x] Development: Add README and CHANGELOG
- [x] Section about development / contributions
- [x] Add and update CONTRIBUTORS
- [x] Testimonials => Gallery. Fix links to https://vmm.panodata.net/
- [x] –help => Usage
- [x] Refer to PostgreSQL “trust”-based authentication
- [x] Change copyright name
- [x] Add `sphinx-copybutton` and `sphinx-tabs`
- [x] Remove version number at left top?
- [/] Trim left-hand menu
- [x] Copyright year
- [x] Other projects: Add SCxxx
- [x] Interlink with forum
- [/] Use sphinx-inline-tabs
- [x] Add sphinxext-opengraph
- [x] Improve gallery
- [x] Add a bit of eye candy to the landing page

3.3.11 Done

All the machinery

- [x] Download cache for data feed (5 minutes)
- [x] Write metadata directly to Postgres
- [x] Redesign commandline interface
- [x] Create CHANGES.rst, update documentation and repository (tags)
- [x] Add tooling for packaging
- [x] Publish to PyPI
- [x] Write measurement data directly to InfluxDB

- [x] Store stations / data **while** processing
- [x] Make a sensor type chooser in Grafana. How would that actually select multiple(!) stations by id through Grafana?
- [x] Store Geohash into InfluxDB database again. Check for sensor_id.
- [x] Probe Redis when starting
- [x] Add Grafana assets
- [x] Import historical data from http://archive.luftdaten.info/
- [x] Check User-Agent settings
- [x] Overhaul station metadata process:
  1. Collect station information from API or CSV into PostgreSQL
  2. Export station information from PostgreSQL as JSON, optionally in format suitable for Panodata Map Panel.
- [x] Improve README
  - [x] Add link to Demo #5
  - [x] Mention InfluxDB storage and historical data
  - [x] Add some screenshots
- [x] Add more sensors:
  - archive.luftdaten.info/2017-10-08/2017-10-08_pms3003_sensor_366.csv
  - archive.luftdaten.info/2017-10-08/2017-10-08_pms7003_sensor_5920.csv
  - archive.luftdaten.info/2017-11-25/2017-11-25_hpm_sensor_7096.csv
  - archive.luftdaten.info/2017-11-26/2017-11-26_bmp280_sensor_2184.csv
  - archive.luftdaten.info/2017-11-26/2017-11-26_htu21d_sensor_2875.csv
- [x] Speed up CSV data import using UDP?
- [x] Add PostgreSQL view "ldi_view" with ready-computed name+station_id things and more
- [x] Improve RDBMS database schema
  - [x] Rename "weatherbase" to "weatherbase"
  - [x] Rename id => station_id
  - [x] Rename osm => osm_*
  - [x] Rename ldi_view => ldi_network
- [x] Fix Grafana vt+kn exports
- [x] Overhaul Grafana dashboards
- [x] Display number of sensors per family
- [x] Remove –help from README
- [x] Improve README re. setup
- [x] Entrypoints for rendering Grafana JSONs
- [x] New sensor type DS18B20, e.g. `WARNING: Skip import of /var/spool/archive.luftdaten.info/2019-01-01/2019-01-01_ds18b20_sensor_11301.csv. Unknown sensor type`

- [x] Add station_id to “choose multiple stations” chooser
- [x] Add GRANT SQL statements and bundle with “–create-view” to “–setup-database”
- [x] Progressbar for emitting data to target subsystems
- [x] Data plane refactoring
  - Put “sensor_id” into “data/reading” item
  - Streamline processing of multiple readings

More

- [x] Fixed:

```
2019-01-21 02:54:44,787 [luftdatenpumpe.core           ] WARNING: Could not make␣
↪reading from {'sensordatavalues': [{'value': '81.40', 'value_type': 'humidity',
↪'id': 5790214143}, {'value': '0.20', 'value_type': 'temperature', 'id':␣
↪5790214142}], 'sensor': {'sensor_type': {'name': 'DHT22', 'manufacturer': 'various
↪', 'id': 9}, 'pin': '7', 'id': 19755}, 'timestamp': '2019-01-21 01:50:56', 'id':␣
↪2724801826, 'location': {'longitude': '', 'latitude': '47.8120', 'altitude': '58.0
↪', 'country': 'DE'}, 'sampling_rate': None}.
Traceback (most recent call last):
  File "/opt/luftdatenpumpe/luftdatenpumpe/core.py", line 230, in request_live_data
    reading = self.make_reading(item)
  File "/opt/luftdatenpumpe/luftdatenpumpe/core.py", line 290, in make_reading
    self.enrich_station(reading.station)
  File "/opt/luftdatenpumpe/luftdatenpumpe/core.py", line 308, in enrich_station
    station.position.geohash = geohash_encode(station.position.latitude, station.
↪position.longitude)
  File "/opt/luftdatenpumpe/luftdatenpumpe/geo.py", line 351, in geohash_encode
    geohash = geohash2.encode(float(latitude), float(longitude))
TypeError: float() argument must be a string or a number, not 'NoneType'
```

- [x] Spotted this:

```
    2019-01-23 16:08:45,230 [luftdatenpumpe.core           ] WARNING: Could not␣
↪make reading from {'location': {'latitude': 48.701, 'longitude': 9.316},
↪'timestamp': '2018-11-03T02:51:15', 'sensor': {'sensor_type': {'name': 'BME280'},
↪'id': 17950}}.
    Traceback (most recent call last):
      File "/home/elmyra/develop/luftdatenpumpe/lib/python3.5/site-packages/
↪luftdatenpumpe/core.py", line 510, in csv_reader
        if not self.csvdata_to_reading(record, reading, fieldnames):
      File "/home/elmyra/develop/luftdatenpumpe/lib/python3.5/site-packages/
↪luftdatenpumpe/core.py", line 538, in csvdata_to_reading
        reading.data[fieldname] = float(value)
    ValueError: could not convert string to float: '985.56 1541213415071633'

    2019-01-23 16:08:45,282 [luftdatenpumpe.core           ] WARNING: Could not␣
↪make reading from {'location': {'latitude': 48.701, 'longitude': 9.316},
↪'timestamp': '2018-11-03T08:52:15', 'sensor': {'sensor_type': {'name': 'BME280'},
↪'id': 17950}}.
    Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
      File "/home/elmyra/develop/luftdatenpumpe/lib/python3.5/site-packages/
↪luftdatenpumpe/core.py", line 510, in csv_reader
        if not self.csvdata_to_reading(record, reading, fieldnames):
      File "/home/elmyra/develop/luftdatenpumpe/lib/python3.5/site-packages/
↪luftdatenpumpe/core.py", line 538, in csvdata_to_reading
        reading.data[fieldname] = float(value)
    ValueError: could not convert string to float: '985.97 1541235075187801'

Update: Seems to work already, see ``luftdatenpumpe readings --network=ldi --
↪sensor=17950 --reverse-geocode``.
```

IRCELINE

- [x] Add IRCELINE SOS data plane
- [x] Add IRCELINE SOS to Grafana and documentation
- [x] Add filtering for SOS API, esp. by station id
- [x] Add time control, date => start, stop parameters or begin/end
- [x] Fix slugification of IRCELINE name “wind-speed-scalar-”
- [x] Ignore `--country=BE` when operating on IRCELINE

3.4 Data models for luftdaten.info network

3.4.1 API and data schema

Please have a look at the upstream documentation first:

- https://github.com/opendata-stuttgart/meta/wiki/APIs
- https://github.com/opendata-stuttgart/meta/wiki/EN-APIs

schema has more information and example documents about LDI’s official data schema in JSON format.

3.4.2 Upstream

```
http https://api.luftdaten.info/static/v1/data.json | \
    jq '[ .[] | select(.location.id==49 or .location.id==1033) ]' > upstream.json
```

3.4.3 Downstream

Running Luftdatenpumpe without reverse geocoding yields compact output, while enabling geocoding will enrich the station information significantly.

Stations

```
# No reverse geocoding.
luftdatenpumpe stations --network=ldi --station=49,1033 > stations-compact.json

# With reverse geocoding.
luftdatenpumpe stations --network=ldi --station=49,1033 --reverse-geocode > stations-
↪geocoded.json
```

Readings

```
# No reverse geocoding.
luftdatenpumpe readings --network=ldi --station=49,1033 > readings-compact.json
```

```
# With reverse geocoding.
luftdatenpumpe readings --network=ldi --station=49,1033 --reverse-geocode > readings-
↪geocoded.json
```

3.5 Data models for IRCELINE network

3.5.1 Upstream

Stations

```
# Stations
http 'http://geo.irceline.be/sos/api/v1/stations?expanded=true' | \
    jq '[ .[] | select(.properties.id==1234 or .properties.id==1720) ]' \
    > upstream-stations.json
```

Readings

```
# Readings
http POST http://geo.irceline.be/sos/api/v1/timeseries/getData \
    timeseries:='[10744,7185]' \
    timespan='PT3h/2019-04-24T01:00:00Z' | \
    jq . > upstream-timeseries.json
```

3.5.2 Downstream

Running Luftdatenpumpe without reverse geocoding yields compact output, while enabling geocoding will enrich the station information significantly.

Stations

```
# No reverse geocoding.
luftdatenpumpe stations --network=irceline --station=1234,1720 > stations-compact.json

# With reverse geocoding.
luftdatenpumpe stations --network=irceline --station=1234,1720 --reverse-geocode >␣
↪stations-geocoded.json
```

Readings

```
# No reverse geocoding.
luftdatenpumpe readings --network=irceline --station=1234,1720 > readings-compact.json

# With reverse geocoding.
luftdatenpumpe readings --network=irceline --station=1234,1720 --reverse-geocode >␣
↪readings-geocoded.json
```

CHAPTER

FOUR

RESEARCH

4.1 Air quality information - open data sources

4.1.1 European Environment Agency

- https://www.eea.europa.eu/themes/air/air-quality-index

4.1.2 Netherlands

KNMI / RIVM

Sensor Observation Service (SOS) and data management for Air Quality data from RIVM.

- https://www.ru.nl/nsm/imr/our-research/departments/geography-planning-environment/projects/smart-emission-project/
- https://sospilot.readthedocs.io/en/latest/data.html
- https://sospilot.readthedocs.io/en/latest/weather.html
- https://github.com/Geonovum/sospilot/issues/17
- https://inspire.rivm.nl/sos/eaq/service?service=SOS&request=GetCapabilities

Smartplatform

- https://smartplatform.readthedocs.io/
- https://buildmedia.readthedocs.org/media/pdf/smartplatform/latest/smartplatform.pdf
- https://web.archive.org/web/20210724005627/https://justobjects.nl/category/smartemission/
- https://justobjects.nl/emit-6-airsenseur-calibration/
- https://smartemission.ruhosting.nl/
- https://web.archive.org/web/20201129035112/https://data.smartemission.nl/
- https://web.archive.org/web/20180531165312/http://data.smartemission.nl/smartapp/
- https://web.archive.org/web/20180602065938/http://data.smartemission.nl/heron/
- https://heron-mc.org/
- https://smartplatform.readthedocs.io/
- `https://data.smartemission.nl/grafana/d/kHCobhViz/allsensordata`

- https://data.smartemission.nl/grafana-dc/d/HVSBmbHmz/airsenseur-netherlands-deploy
- https://data.smartemission.nl/grafana/
- https://data.smartemission.nl/grafana-dc/

4.1.3 Miscellaneous

- https://community.hiveeyes.org/t/ircel-celine-das-feinstaubmessnetzwerk-der-belgischen-interregionalen-umweltagentur/1271
- https://community.hiveeyes.org/t/luchtmeetnet-dutch-air-quality-networks/1272
- https://community.hiveeyes.org/t/uk-air-air-information-resource/1273
- https://community.hiveeyes.org/t/blume-feinstaub-daten-api-im-sos-standard/1277
- https://community.hiveeyes.org/t/prevair-air-information-ressource-in-france/1294
  - https://www.data.gouv.fr/fr/reuses/pollution-de-lair-dans-les-ecoles/
  - https://www.respire-asso.org/pollution-de-lair-dans-les-ecoles/
- https://community.hiveeyes.org/t/weather2stats-processes-data-from-different-weather-data-sources/1370

4.1.4 Weather data

- https://darksky.net/
- https://github.com/earthobservations/wetterdienst

4.2 What others are doing

4.2.1 BLUME

- https://community.hiveeyes.org/t/blume-messnetz-api/1234
- https://www.codefor.de/projekte/2015-08-08-be-feinstaub-sos.html
- https://github.com/52North/js-sensorweb-client
- https://www.codefor.de/projekte/2014-07-01-be-blume.html
- https://github.com/dirkschumacher/blume_messnet_api
- https://github.com/dirkschumacher/blume_crawler
- https://github.com/johnjohndoe/blume_sos_adapter
- https://github.com/52North/sos-importer
- https://github.com/johnjohndoe/SOS-Importer-Blume

4.2.2 OpenAQ

- https://openaq.org/
- https://github.com/openaq
- https://github.com/dolugen/openaq-browser
- https://docs.openaq.org/
- https://dolugen.github.io/openaq-browser/

4.2.3 openHAB weather bindings

- https://www.openhab.org/addons/bindings/weather1/
- https://community.openhab.org/t/weather-widget/45437
- https://community.openhab.org/t/solved-json-path-weather-warnings-dwd-deutscher-wetterdienst/46295

4.2.4 OpenSense

- https://www.opensense.network/
- https://github.com/opensense-network
- https://twitter.com/sallapf/status/1070334518106750977

4.2.5 Python packages

- https://github.com/search?q=luftdaten&type=Repositories
- https://pypi.org/search/?q=luftdaten
- https://github.com/ieservices/luftdaten-big-data-analysis.info
- https://github.com/calderacc/luft-jetzt
- https://github.com/dataunity/luftdaten
- https://github.com/fabaff/python-luftdaten
- https://pypi.org/project/airqdata/
- https://github.com/dr-1/airqdata

4.2.6 QuantAQ

- https://www.quant-aq.com/
- https://github.com/quant-aq

4.2.7 Sensor.Community public data aggregator

- https://github.com/pjgueno/SCPublicData
- http://pjgueno.epizy.com/SCPublicData/
- https://forum.sensor.community/t/scraping-pm-data-help-needed/1448

4.2.8 Miscellaneous

- http://www.ise.tu-berlin.de/fileadmin/fg308/publications/2018/borges_et_al_open_enviromental_data.pdf
- https://aircitizen.org/en/
- https://www.openmaps.online/LuftdatenInfoTimeslider/
- https://github.com/OpenDEM/LuftdatenInfoTimeslider
- https://github.com/luftdata
- https://luftdata.se/
- https://plugins.qgis.org/plugins/SOSClient/
- https://github.com/ruben-mv/SOSClient/blob/master/sos/sos.py
- https://pypi.org/project/OWSLib/
- https://geopython.github.io/OWSLib/
- https://github.com/geopython/OWSLib/blob/master/owslib/sos.py
- https://geopython.github.io/OWSLib/usage.html#sos-2-0
- https://github.com/peterataylor/OWSLib-SOS-example
- https://nbviewer.jupyter.org/github/peterataylor/OWSLib-SOS-example/blob/master/owslib-sos-notebook.ipynb

4.3 Tech Radar

4.3.1 References

luftdaten.info

- https://web.archive.org/web/20220604103954/https://luftdaten.info/
- https://sensor.community/
- https://archive.luftdaten.info/
- https://deutschland.maps.luftdaten.info/

Resources

- opendata-stuttgart/sensors-software: Support for InfluxDB and MQTT as backend.
- https://getkotori.org/docs/applications/luftdaten.info/
- https://community.hiveeyes.org/t/datenmischwerk/702
- https://community.hiveeyes.org/t/environmental-metadata-library/1190
- https://community.hiveeyes.org/t/erneuerung-der-luftdatenpumpe/1199
- https://community.hiveeyes.org/t/ldi-dataplane-v2/1412

SE Data Platform

- https://web.archive.org/web/20201129035112/https://data.smartemission.nl/
- https://smartplatform.readthedocs.io/
- https://smartplatform.readthedocs.io/en/latest/api.html
- https://github.com/smartemission/docker-se-stetl
- https://justobjects.nl/
- https://github.com/Geonovum
- https://www.geonovum.nl/themas/testbed-ogc-apis
- https://www.geonovum.nl/themas/kennisplatform-apis
- https://osgeo.nl/
- https://geotoko.nl/
- https://nlextract.nl/
- https://github.com/nlextract/NLExtract

JRC

- https://publications.jrc.ec.europa.eu/repository/bitstream/JRC102703/jrc102703%20-1600907%20host%20report%20final.pdf
- https://inspire.ec.europa.eu/sites/default/files/presentations/observational_and_measurement_data_in_inspire_-_inspire_conference_2017-final_1.pdf
- https://www.slideshare.net/justb4/open-sensor-networks-with-lora-ttn-and-sensorthings-api
- https://smartplatform.readthedocs.io/en/latest/_static/dissemination/rivm-17jan2017/SmartEmission-RIVM-170117.pdf

Technologies

Standing on the shoulders of giants.

Databases

- https://github.com/influxdata/influxdb
- https://dataset.readthedocs.io/
- https://www.sqlalchemy.org/
- https://www.postgresql.org/
- https://postgis.net/
- https://github.com/pramsey/pgsql-http
- https://redis.io/

Software and services

- https://github.com/grafana/grafana
- https://grafana.com/grafana/plugins/grafana-worldmap-panel/
- https://en.wikipedia.org/wiki/Geohash
- https://nominatim.org/