Grafana Dashboard for Phenological Data Monitoring

1 Introduction

This project aims to provide a software architecture of a web-based application that collects phenological data across Germany using open data from Deutscher Wetterdienst (DWD). Phenological data is of great interest to researchers in the field of ecological monitoring, specifically because it can serve as a monitor for climate change [Sch14] is has become more important in recent years.

The developed solution can help visualize phenological data more intuitively using a map of the country, helping researchers judge the current status of plants all around the country at a glance. In the future, this dashboard can be combined with another data-source to conduct more objective-specific research, i.e. the correlation of beehive data with phenological data in a given location.

The project consists mainly of two parts:

- Back-end, which consists of an application written in Python that collects data from DWD daily. After that it stores new data in the database;
- Front-end is done by leveraging the Grafana open-source dashboard solution.

2 Current developments and state-of-the-art

This chapter gives a brief review of previous attempts at building systems that are similar in spirit. After that follows a brief review of the state-of-the-art technologies which are currently available and can aid in building this type of project.

2.1 Current developments

With the advancements and affordability of IoT sensor devices, there has been an emergence of sensor data aggregation and processing projects. One of such projects is luftdaten.de (Sensor Community)[1], which collects widely available data such as air temperature, humidity, etc. from the volunteers in numerous countries. They do not have a dashboard; however, they provide a map to visualize recently collected data (no history retention).

Another ambitious project in the area of ecological data aggregation applied specifically to beehives is called Hiveeyes[2], which focuses on “developing a flexible beehive

[1] Sensor Community. URL: https://sensor.community
[2] The Hiveeyes Project. URL: https://hiveeyes.org

monitoring infrastructure platform" by providing an open-source toolkit for non-invasive beekeeping. This allows beekeepers to get a head start in building data-gathering infrastructure and visualization complex. It uses the Grafana dashboard as the main way to build visualizations.

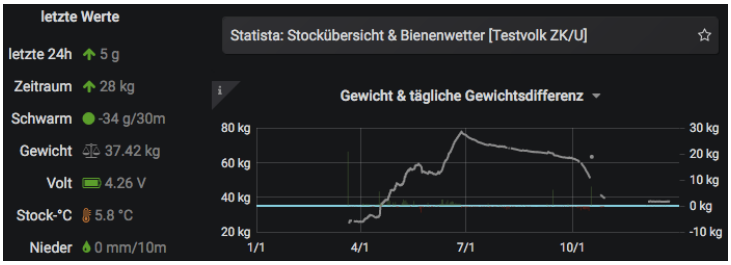


Fig. 1: Example Hiveeyes dashboard

Deutscher Wetterdienst (DWD) is a German meteorological service that provides weather data and forecasting around Germany. Alongside weather services it also provides other types of data for nautical, aviation, and agricultural purposes. In particular, it provides such data as min, max and mean temperature, historical precipitation, atmospheric pressure, cloud coverage, wind force, etc.

DWD provides its data through the Climate Data Center (CDC)[3] that consists of a portal, which allows the user to get interactive access to data set visualizations (Fig. 2).

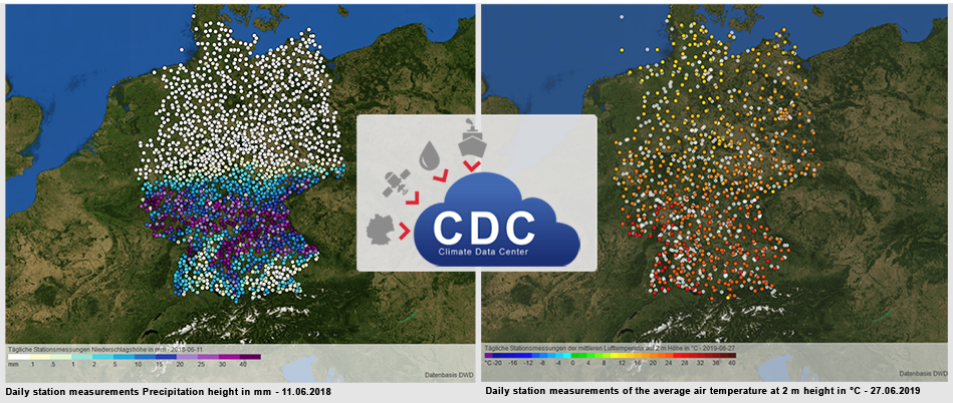
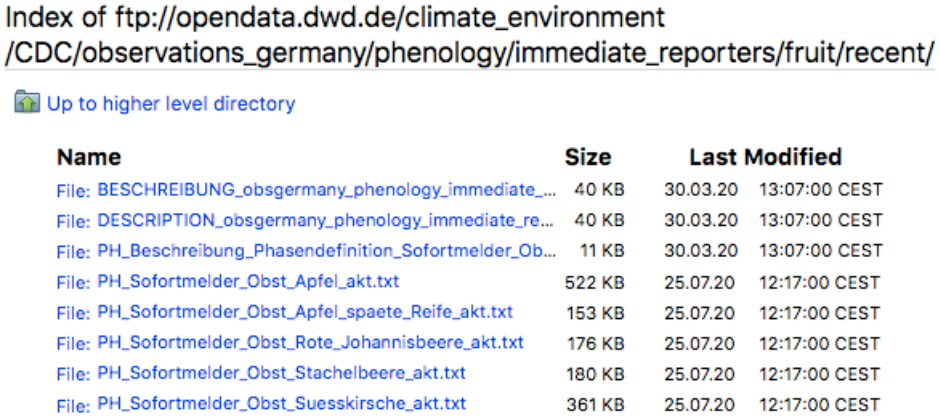


Fig. 2: CDC-portal data set visualization using an interactive map interface

[3] Deutscher Wetterdienst Climate Data Center. URL: https://www.dwd.de/EN/climate_environment/cdc/cdc_node.html

Another way to access DWD data is through the CDC OpenData[4] repository that provides access to the larger variety of data that is accessible through either HTTPS or FTP. Among the different kinds of data sets provided one of the most interesting for this project is the phenological data set, which is accessible from the repository.


Index of ftp://opendata.dwd.de/climate_environment
/CDC/observations_germany/phenology/immediate_reporters/fruit/recent/

Up to higher level directory

Name	Size	Last Modified
File: BESCHREIBUNG_obsgermany_phenology_immediate_...	40 KB	30.03.20 13:07:00 CEST
File: DESCRIPTION_obsgermany_phenology_immediate_re...	40 KB	30.03.20 13:07:00 CEST
File: PH_Beschreibung_Phasendefinition_Sofortmelder_Ob...	11 KB	30.03.20 13:07:00 CEST
File: PH_Sofortmelder_Obst_Apfel_akt.txt	522 KB	25.07.20 12:17:00 CEST
File: PH_Sofortmelder_Obst_Apfel_spaete_Reife_akt.txt	153 KB	25.07.20 12:17:00 CEST
File: PH_Sofortmelder_Obst_Rote_Johannisbeere_akt.txt	176 KB	25.07.20 12:17:00 CEST
File: PH_Sofortmelder_Obst_Stachelbeere_akt.txt	180 KB	25.07.20 12:17:00 CEST
File: PH_Sofortmelder_Obst_Suesskirsche_akt.txt	361 KB	25.07.20 12:17:00 CEST

Fig. 3: CDC OpenData phenological data repository for fruit plants

Data is stored as plain text CSV files, updated daily, and categorized by plant type (crops, fruit, and wild plants). This method is much more versatile since it gives access to more types of data for researchers at a cost of extra work processing data.

Since DWD maintains the biggest network of stations all around Germany and provides open phenological data with frequent updates, it makes sense to use it as a data source for this project.

2.2 State-of-the-art

The next part of this chapter gives a brief review of the state-of-the-art open-source tools and solutions for data processing and visualization.

Since this project aims to build a dashboard graphical user interface for phenological data visualization it makes sense to use already existing tools that provide such functionality.

Grafana[5] is an open-source web application for analytics and interactive data visualization using charts, graphs, tables, and even maps – which are particularly useful for this project. Grafana allows to connect various types of data sources, such as time-series databases, logging, and even SQL relational databases such as MySQL and PostgreSQL.

[4] CDC OpenData. URL: https://opendata.dwd.de/climate_environment/CDC
[5] Grafana: The open observability platform | Grafana Labs. URL: https://grafana.com

By adding panels, it allows users to build their own dashboard, where each panel displays a particular query to the specified data source.

Fig. 4: Grafana open-source dashboard

An alternative to Grafana could serve InfluxDB – which originally was built as a time-series database, but now also features a dashboard-building web application called Chronograf. However, it is aimed more towards data exploration and tight integration with InfluxDB, which makes it less than ideal choice for this project, since the database that is going to be used in this project (MySQL) is not supported along with missing geographic map visualization.

As have been previously mentioned this project will use MySQL relational database for storage of normalized phenological data, gathered from the DWD CDC OpenData repository. This will allow for fast access to data since it is possible to leverage indexing. MySQL was chosen since it is one of the most popular and supported open-source databases and Grafana already comes with data source integration for this database.

To prepare data for insert into a relational database first it must be downloaded from the DWD repository and then processed. One of the commonly used solutions for data processing is the Python[6] programming language using pandas[7] library, which makes it easy to work with table data.

[6] Python programming language. URL: https://www.python.org
[7] pandas - Python Data Analysis Library. URL: https://pandas.pydata.org

3 Application architecture

The application consists of two main parts: front-end and back-end. Front-end will be implemented using the Grafana web application, setting up a data source, and configuring the dashboard by adding panels to it.

The back-end is where the most work in this project is being done – it involves not only building an application but also designing a data model and coming up with a methodology for data transformation to represent phenological data intuitively.

3.1 Data flow

The amount and kind of data affect the implementation of the application. To understand the structure of the application better, it is useful to the flow of data. DWD CDC OpenData repository provides plant text CSV files that contain phenological records for various plants, further put into one of the three categories: crops, fruits, or wild.

Here is shown an excerpt from the file *PH_Sofortmelder_Obst_Apfel_akt.txt*[8] that contains historical records for apple tree phenological data where each record references the station measurement was taken at, object id (310 for apple tree), phase id, and the date of record.

```
Stations_id; ... Objekt_id; Phase_id; Eintrittsdatum;
        140; ...       310;       29;       20170806;
        140; ...       310;        5;       20180425;
        140; ...       310;       29;       20180729;
```

The data is somewhat normalized already since instead of station, plant, and phase data there are ids which are described in other files:

- PH_Beschreibung_Phaenologie_Stationen_Sofortmelder.txt
- PH_Beschreibung_Pflanze.txt
- PH_Beschreibung_Phase.txt

Because of the nature of the data, it is possible to store it in a normalized fashion using a relational database such as MySQL. A potential application data flow diagram is shown in Fig. 5.

[8] Apple tree phenological data. URL: https://opendata.dwd.de/climate_environment/CDC/observations_germany/phenology/immediate_reporters/fruit/recent/PH_Sofortmelder_Obst_Apfel_akt.txt

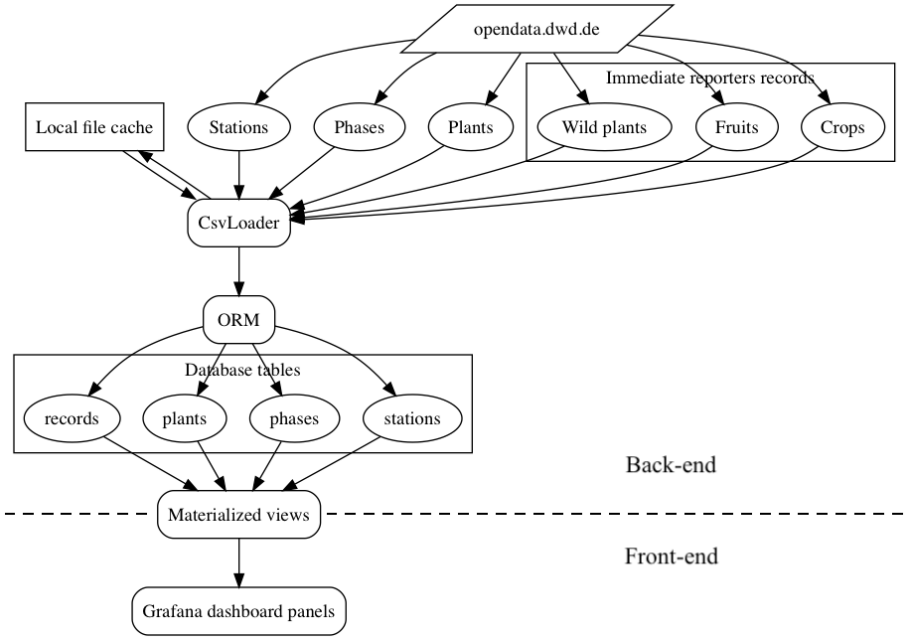


Fig. 5: Application data flow

On the figure above the data from *opendata.dwd.de* service is downloaded using some data loader over HTTP. Since some of the data, such as information about the stations, phases, and plant names does not change often it is useful to store it locally in a file cache. After the data has been downloaded and processed it should be pushed to the database. A common way to do this is by using an object-relational mapping (ORM), which helps to abstract database-specific language and details by the means of providing an application programming interface (API) to the developer.

In practice, this also helps with creating a database table schema by defining it in code instead of doing this manually through the database command-line interface (CLI) by typing SQL instructions. Such approach also helps to keep database schema in sync with the code, reducing the number of errors while also abstracting from a specific database. After the data has been pushed to the respective database tables it is useful to generate views for the data in the database, which can give multiple benefits:

1. Increasing security by allowing access from the dashboard to the slice (view) of data instead of all of it;
2. By removing extra data and adding indexes to the fields which are used to filter data, which can significantly increase the speed of queries;
3. It gives the ability to calculate aggregate values of the data (as will further be shown).

3.2 Architecture

Since this is a complex application, it will require multiple components to function:

- MySQL database server;
- Grafana web application;
- Web server to host application assets (i.e. images for the use in the dashboard);
- Software for phenological data download, processing, and upload to the database.

The application will be hosted on a Linux server using Digital Ocean VPS hosting. It is going to be served on the web using the Nginx[9] reverse proxy and web server. For the prototype, all of the components will be hosted on one server. Requests to Nginx are going to be proxied to Grafana, which itself uses an Apache web server on a non-standard HTTP port (3000). On the figure below proposed application infrastructure is shown.

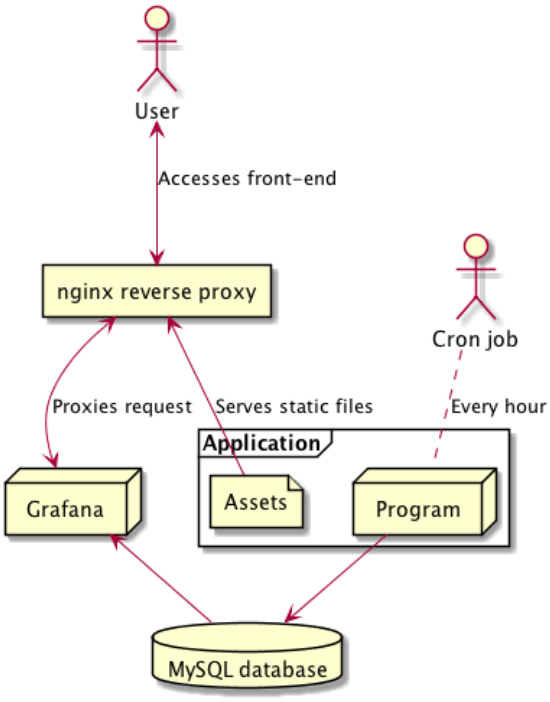


Fig. 6: Application infrastructure

The program itself is run on a schedule using cron job scheduler. Using Nginx as a web server an application assets directory is going to be hosted, which will allow hosting custom files for the dashboard, such as images, along with the application. Grafana and

[9] NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy. URL: https://www.nginx.com

the application will share the database. However, whilst Grafana will be only accessing data in read-only mode through generated materialized views, the application will have full control over the database – having an ability to insert data and change table schema through migrations (discussed later in the implementation section).

4 Methodology: phase formalization and data processing

This chapter discusses how the data from the DWD CDC OpenData repository is processed. To present it in the Grafana dashboard, retrieved data has to be processed and transformed. Since phases in the repository are presented by text description, a formalization procedure has to be carried out to formalize vague text descriptions of phases into a stricter format.

4.1 Phase formalization

There needs to be a method to aggregate phenological phase data effectively – giving an ability to judge at a glance. Fig. 7 shows a heatmap for each phase, where temperature corresponds to the day of the year in specified bounds. This approach helps visualize intuitively locations where the given phase occurs on the given day of the year. However, it has a drawback in that each map represents the distribution of just one phase, not giving a complete overview and ability to compare phases between each other, and because of this such method is not suitable for this project.

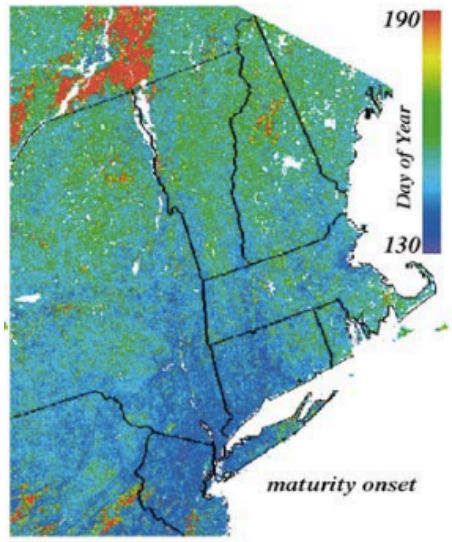


Fig. 7: Location vs day of the year for a given phase. Source: [Zha03]

Since the user has to be able to judge using a single map at a glance about the current phenological phases all around the country an approach should be used that would make it possible to compare phases between each other. One of the ways to do this is to assign a numeric weight to each phase (Tab. 1).

Weight	Phase ID	Name
1	1	beginning of turning green
2	5	beginning of flowering
3	7	end of flowering
4	24	harvest
-1	31	autumn coloring of leaves

Tab. 1: Phase weights depending on their description

This gives an ability to compare phases formally and because of this, it becomes possible to display them on a graph in a variety of formats – for example as colored dots on the map, where the color corresponds to the weight of the phase. This approach is similar to the BBCH scale, which is a scale from 0 to 9 that describes the whole phenological process from the beginning of turning green to ripening and harvest or autumn leaf fall (Fig. 8, Fig. 9).

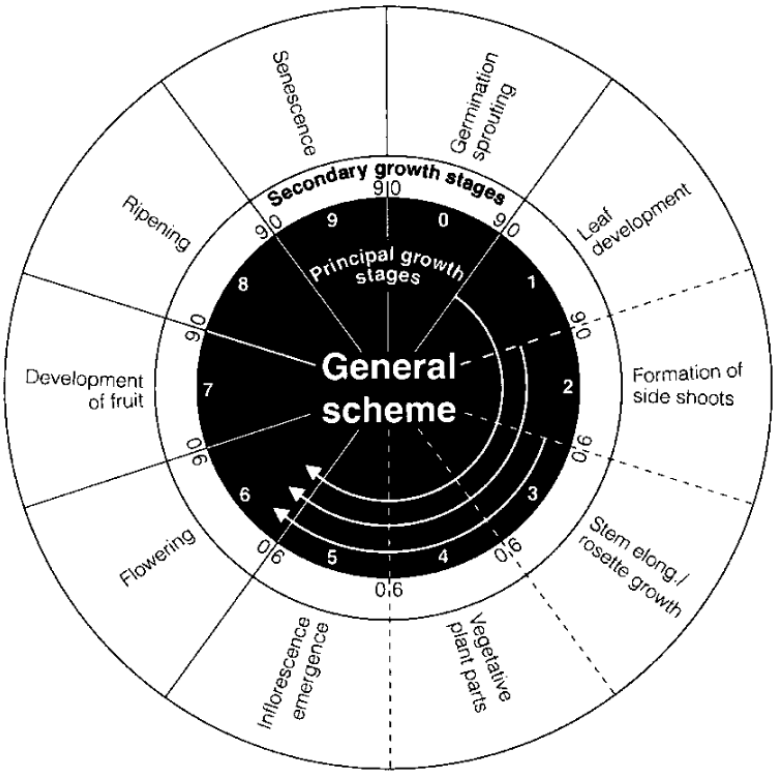


Fig. 8: Main growth stages put on a scale from 0 to 9. Source: [BBC01]

Fig. 9: Phenological growth stages according to the BBCH scale. Source: [Yan19]

By grouping phenological records from DWD CDC OpenData repository by date and station (name and location) and averaging the weights of the phases of all plants in the area for which records exist, it is possible to get insights into the general phenological situation for the given station. Since the data provided is already classified into different plant kinds (crops, fruits, and wild) it makes sense not to keep this grouping, since different kinds of plants might have significantly different phase timing, which in turn might make observation data non-representative.

To verify these previous years' data has been grouped by region and plant type. The resulting dataset was further grouped by date to get time-series from it while aggregating weight values as a sum for each date. The cumulative value was chosen because it is more visible than the average on the graph, however, introducing bias to the data since the cumulative value is dependent on the number of records might not be equal for all stations and dates. The resulting graphs are shown in Fig. 10, which proves that this approach works and hence it will be used in this project.

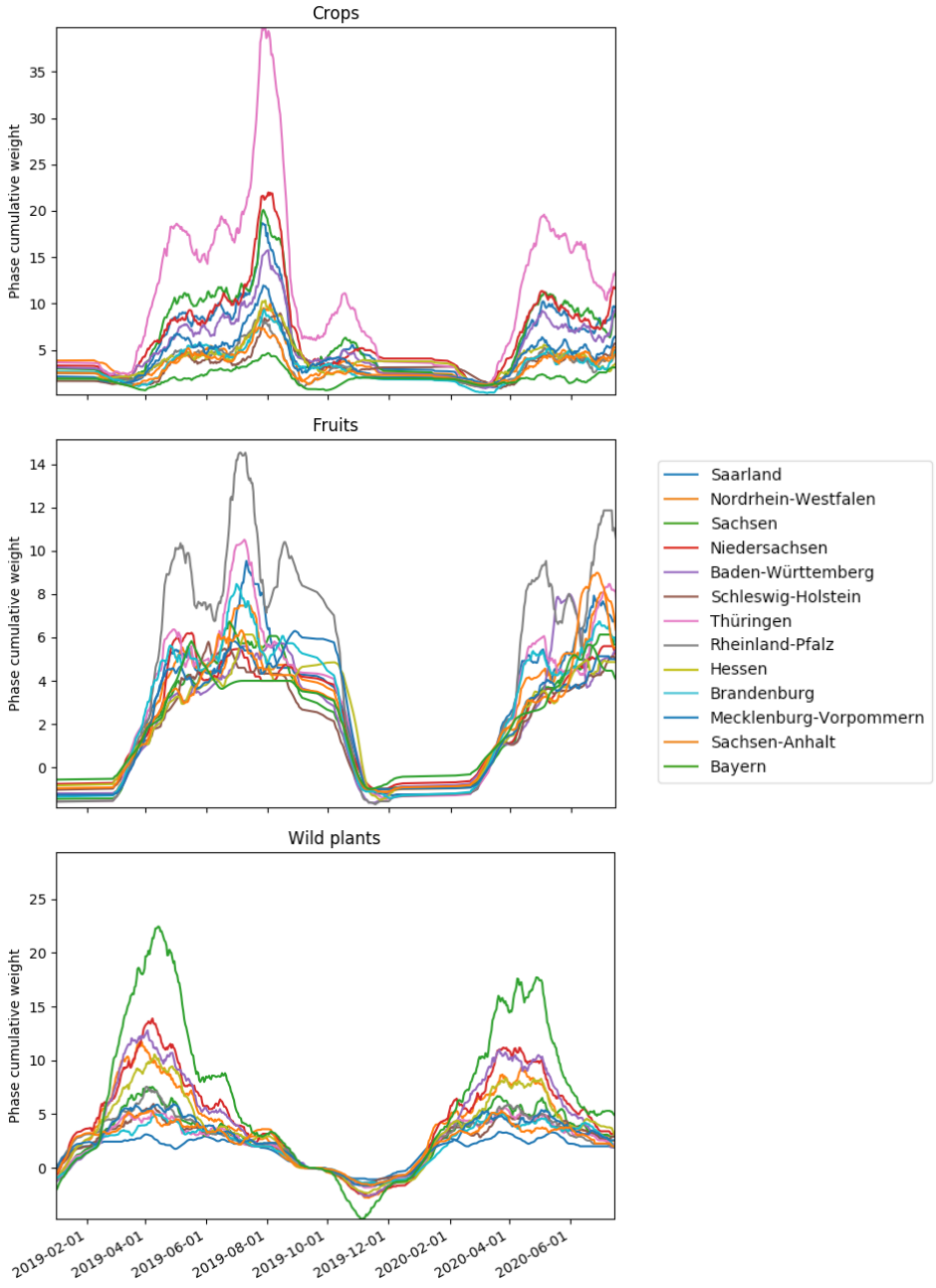


Fig. 10: Phenological phases cumulative weight 30-day moving average by region

4.2 Data processing

Data processing involves cleaning up and processing of raw data that is downloaded from the DWD CDC OpenData repository, namely:

- Converting non-standard CSV format into a table representation inside the program;
- Normalizing column naming scheme, since original data contains non-uniform column table conversion with columns in some documents having a different case. This step also allows to enforce stricter and application-specific naming conversion;
- Stripping whitespaces surrounding the data;
- Extraction and conversion to application data types, since all data in raw CSV files, is stored as text.

Here is an excerpt from the phase data definition file (original whitespaces preserved). It is obvious that this file does not follow CSV standard: it contains semicolons instead of commas as separators as well as each row includes an extra column named “eor” which denotes the end of the row, even though the file itself already contains a newline symbol.

```
Stations_id; … Objekt_id; Phase_id; Eintrittsdatum; … eor;
        140; … 310;             29;       20170806; … eor;
        140; … 310;              5;       20180425; … eor;
```

After parsing this file, it is converted into a pandas data frame, which is a table representation of the data in a Python program. During parsing, whitespaces are removed. Then columns are modified in the program: column is renamed using a specified map. Before this column names are converted to lower case because case varies in different files provided by DWD.

```
{
  "streams": [
    {
      "type": "dwd-phenology",
      "field_map": {
        "stations_id": "station_id",
        "referenzjahr": "year",
        "qualitaetsniveau": "data_quality_bit",
        "objekt_id": "object_id",
        "phase_id": "phase_id",
        "eintrittsdatum": "date",
        "eintrittsdatum_qb": "date_quality_bit",
        "jultag": "day_of_year"
      }
    }
  ]
}
```

The resulting pandas data frame is shown below:

```
      station_id … object_id  phase_id        date …
0             140 …       310        29  2017-08-06 …
1             140 …       310         5  2018-04-25 …
```

The same procedure is done with stations, phases, and plants. However, before pushing phases and plants their names are extracted to put them in different tables. This allows to fully normalize data.

4.3 Data normalization

Data normalization allows to reduce duplication and facilitate data integrity (avoid missing entries and provide a single source of truth). In this project, plant and phase data can be normalized by extracting their names into separate tables. This will allow to decouple data, which in turn will make it possible to make queries lighter in cases when they will not need plant or phase names.

Furthermore, this will make it possible to add more data to already defined plants and phases in the future, such as names in different languages, making the application more versatile. Fig. 11 shows the normalized data model, which is derived from the data flow diagram presented in Fig. 5. By creating a database schema using this diagram it will be possible to store processed phenological data from DWD in a relational database.

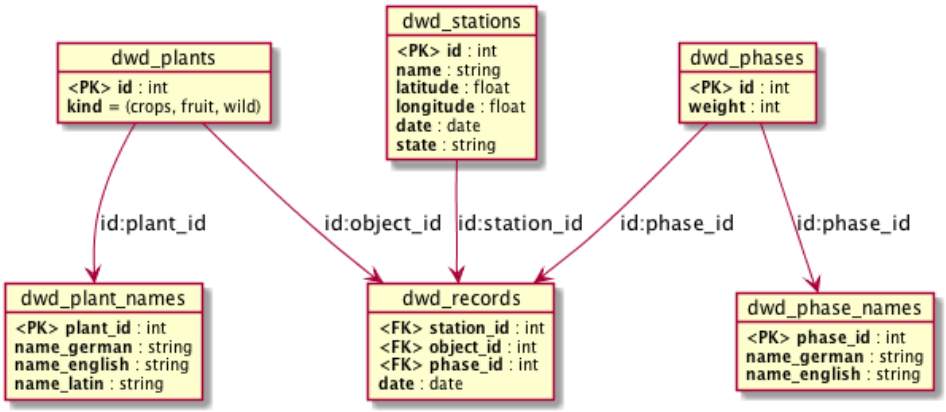


Fig. 11: Normalized data model

5 Implementation

The application is implemented using Python 3.6 programming language. The configuration of the program is stored in a JSON config file (included in the appendix). This file stores configuration for all data streams which are fetched by the program – currently there is only one stream: "dwd-phenology".

5.1 Program

The structure of the project is divided into multiple classes since it is written in Python, which allows to use object-oriented approach. The diagram that represents all core application classes is presented in Fig. 12. It is worth noting that database models and migration classes are not shown on the diagram since they are considered an implementation detail and hence are not important regarding the main purpose of the program (data processing and aggregation).

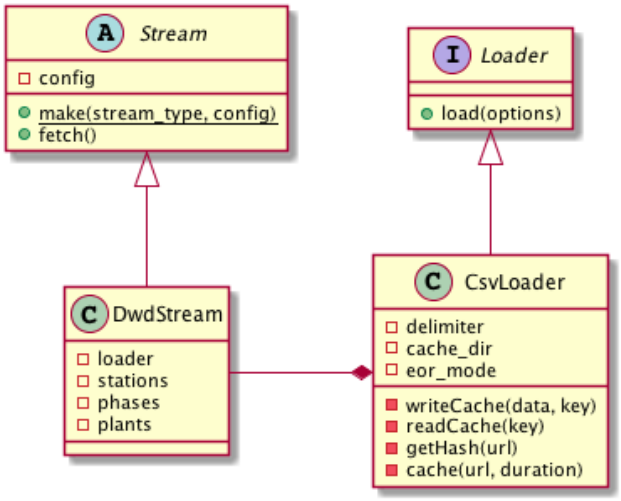


Fig. 12: Simplified application diagram

The diagram above shows the relationship between the stream abstract class which is implemented with the *DwdStream* class that contains all the business logic for data processing and upload to the database. This class has an instance of *CsvLoader*, which is responsible for reading DWD CDC OpenData CSV raw data and returning pandas data frame. After that stream object continues to process downloaded data and pushes it into the database.

5.2 Server

The server is a hosted cloud VPS running Ubuntu 18.04 (Fig. 13). This server is set up to run three software servers: MySQL database server, Grafana application which is served by the Apache webserver on port 3000, and Nginx web server and reverse proxy.

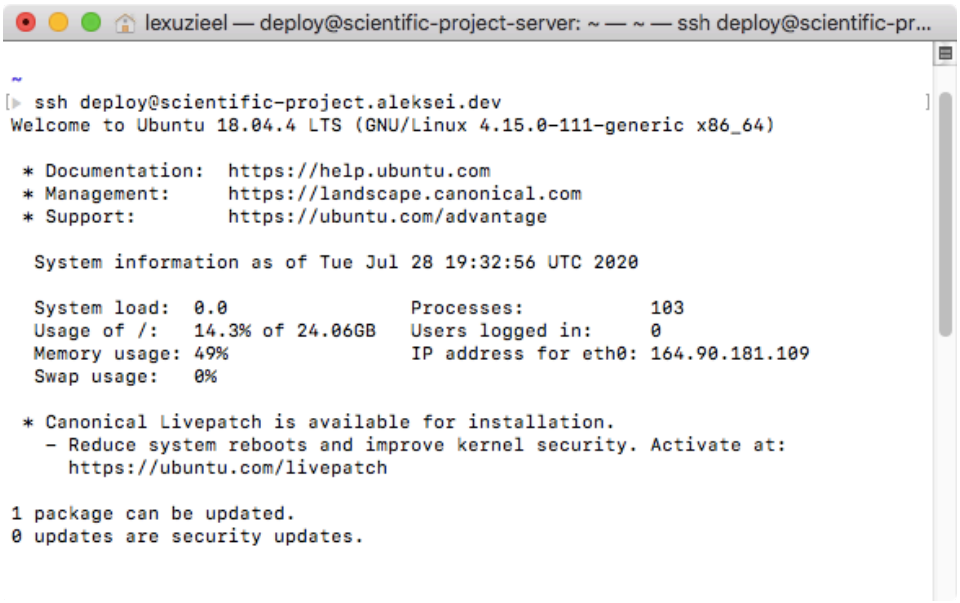


Fig. 13: SSH connection to the Linux VPS

Nginx serves as a reverse proxy to group all of the services and serve all of them via HTTPS using a certificate generated by Let's Encrypt. An excerpt from the Nginx configuration file is shown below:

```
server {
        server_name scientific-project.aleksei.dev;

        location ^~ /data/ {
                alias /home/deploy/app/data/;
        }

        location / {
                proxy_pass http://127.0.0.1:3000;
        }
}
```

The server cron job scheduler configuration file is shown below:

```
# m h  dom mon dow   command
  0 *   *   *   *    bash /home/deploy/app/run.sh
  0 0   *   *   *    bash /home/deploy/app/backup.sh
  * *   *   *   *    cd app && git pull > /dev/null 2>&1
```

5.3 Materialized views

Since MySQL does not have materialized views, they will be imitated by creating and updating cache tables upon each data refresh. Functionally it does not differ from materialized views except for the fact that they will have to be created "manually" from the program each time. This can be easily achieved by using a bash script that will feed an SQL file that defines these tables. Since the program will be run through the bash script by the cron job scheduler anyway this would not inflict any major inconvenience.

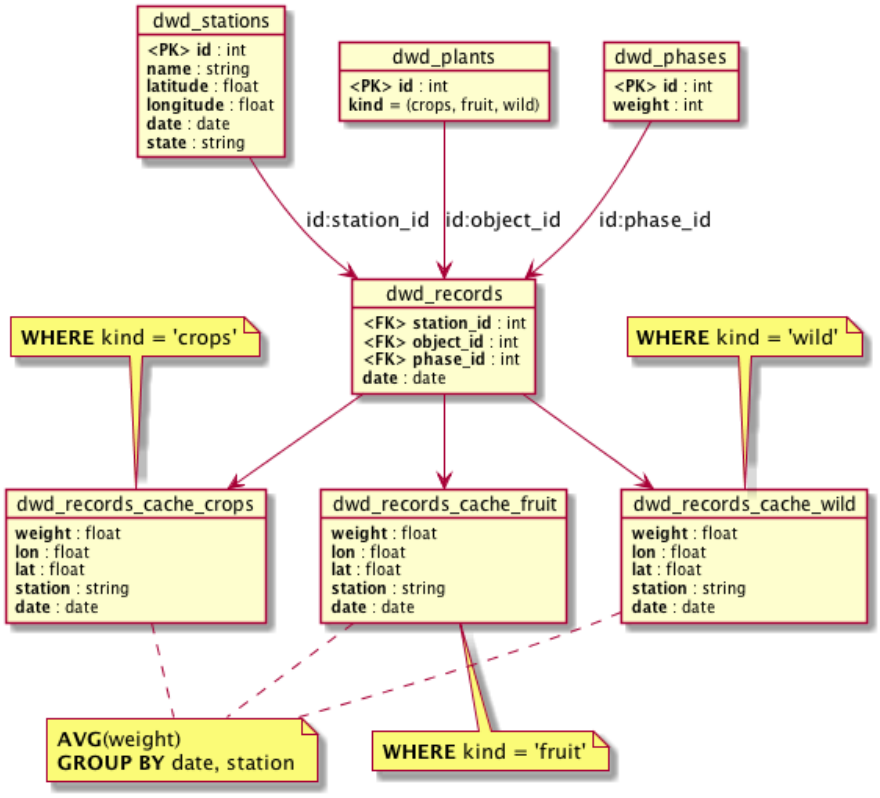


Fig. 14: "Materialized view" cache tables are derived from various tables using SQL JOINs.

5.4 Dashboard creation

The dashboard consists of several panels using the Worldmap Panel plugin[10]. This plugin makes it possible to show data on the map by utilizing the open-source project OpenStreetMap[11].

Fig. 15: Worldmap panel plugin showing data from the database

By utilizing materialized views, it is trivial to build a query in Grafana, since all calculations were already done in the database. Through the addition of indexes on the table fields, it is possible to get a significant speed increase on reading operations, which is exactly what Grafana uses.

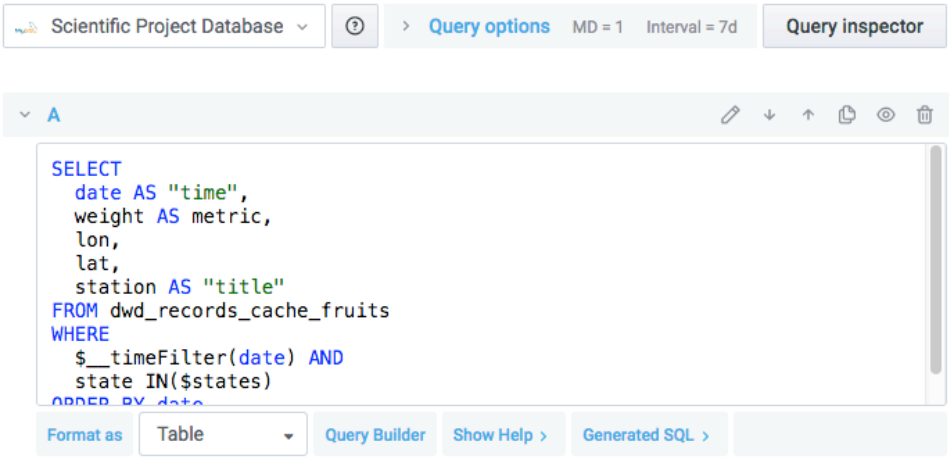


Fig. 16: Defined panel query for fruit plants

A similar procedure is done for other types of plants: crops and wild plants. This results in three panels that each show average phenological phase data for each station location and plant type.

[10] Worldmap Panel plugin for Grafana. URL: https://grafana.com/grafana/plugins/grafana-worldmap-panel
[11] OpenStreetMap. URL: https://www.openstreetmap.org

6 Demonstration

The dashboard is accessible via the link: https://scientific-project.aleksei.dev. Upon loading the user is redirected to the dashboard automatically (Fig. 17). Users can filter the data by regions instead of displaying all the data by using the dropdown menu at the top.

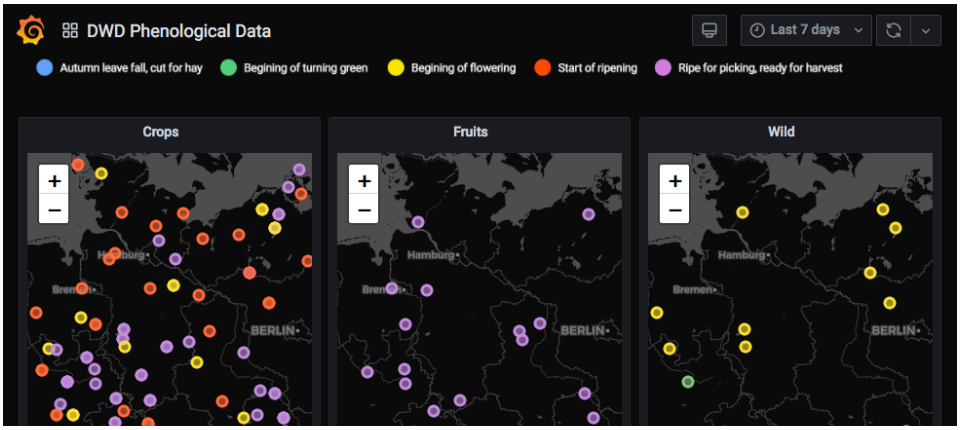


Fig. 17: Grafana dashboard view

The dashboard uses a single data source which is a local MySQL database hosted on the same server (Fig. 18).

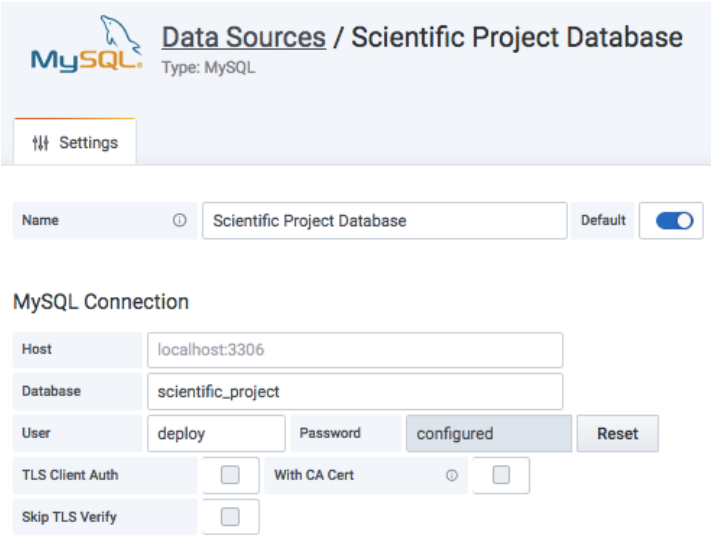


Fig. 18: Grafana data source setup

At the bottom of the dashboard there is a map legend, describing colors with the names of the phases that were given by the DWD:

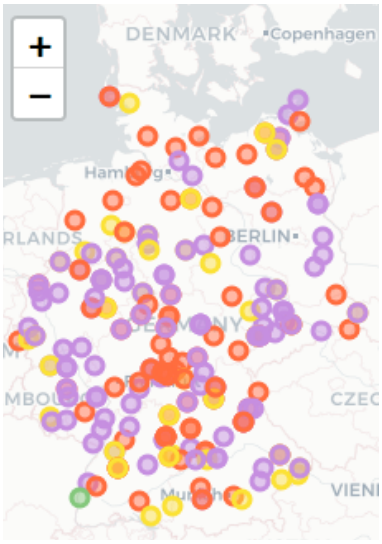


Fig. 19: Crops phases on 2020-07-27

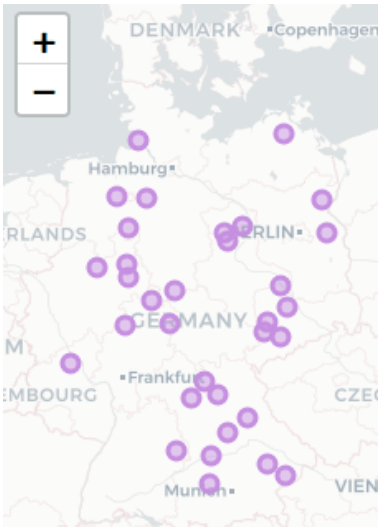


Fig. 20: Fruit phases on 2020-07-27

Color	Name
	autumn leave fall
	autumn colouring of leaves
	second cut for hay
	beginning of turning green
	beginning of unfoalding of leaves
	sprouting of leaves
	general flowering
	beginning of emergence
	beginning of flowering
	tip of tassel visible
	end of flowering
	yellow ripeness
	fruit ripe for picking
	harvest
	grape harvest

Fig. 21: Map legend

7 Conclusion

In this project, an application prototype for phenological data monitoring was developed using Grafana open-source dashboard solution. Open phenological data from the Deutscher Wetterdienst repository was taken and an application was developed to process, and store provided data. To present data in an intuitive fashion, a methodology was developed which allowed to formalize textual phase data and make it possible to visualize it one a map, giving the user a broad overview of the current phenological situation around the country.

The developed methodology resembles BBCH phenological phases scale, however in its current state it could lack the resolution of certain plant types – for example crops, where the last phenological phase appears in the middle of the cycle during cut for hay. This introduces an opportunity to improve the given approach and potentially re-evaluate phase weighting.

Source code for this project is available at:

https://github.com/lexuzieel/dwd-phenology-stream

References

[Sch14] Schmidt G.; Schönrock S. und Schröder W. (2014): Case Study 1: Phenological Trends in Germany. In *Plant Phenology as a Biomonitor for Climate Change in Germany*.

[Zha03] Zhanga X.; Friedla M.A.; Schaafa C.B..S.H. et al. (2003): Monitoring vegetation phenology using MODIS. *Remote Sensing of Environment 84*, 471–475.

[Yan19] Yan-Gang Z.; Gao; Shicai X. et al. (2019): The Phenological Growth Stages of Sapindus Mukorossi According to BBCH Scale. *Forests*, 10, 6.

[BBC01] BBCH working group (2001): *Growth stages of mono-and dicotyledonous plants*. Federal Biological Research Centre for Agriculture and Forestry.