Jetstream - German Wind Data Collector

> Laborarbeit Big Data Engineering WiSe 2024/2025
> Jannis Kaniaros (5934448) & Fabian Lohmüller (8175324)

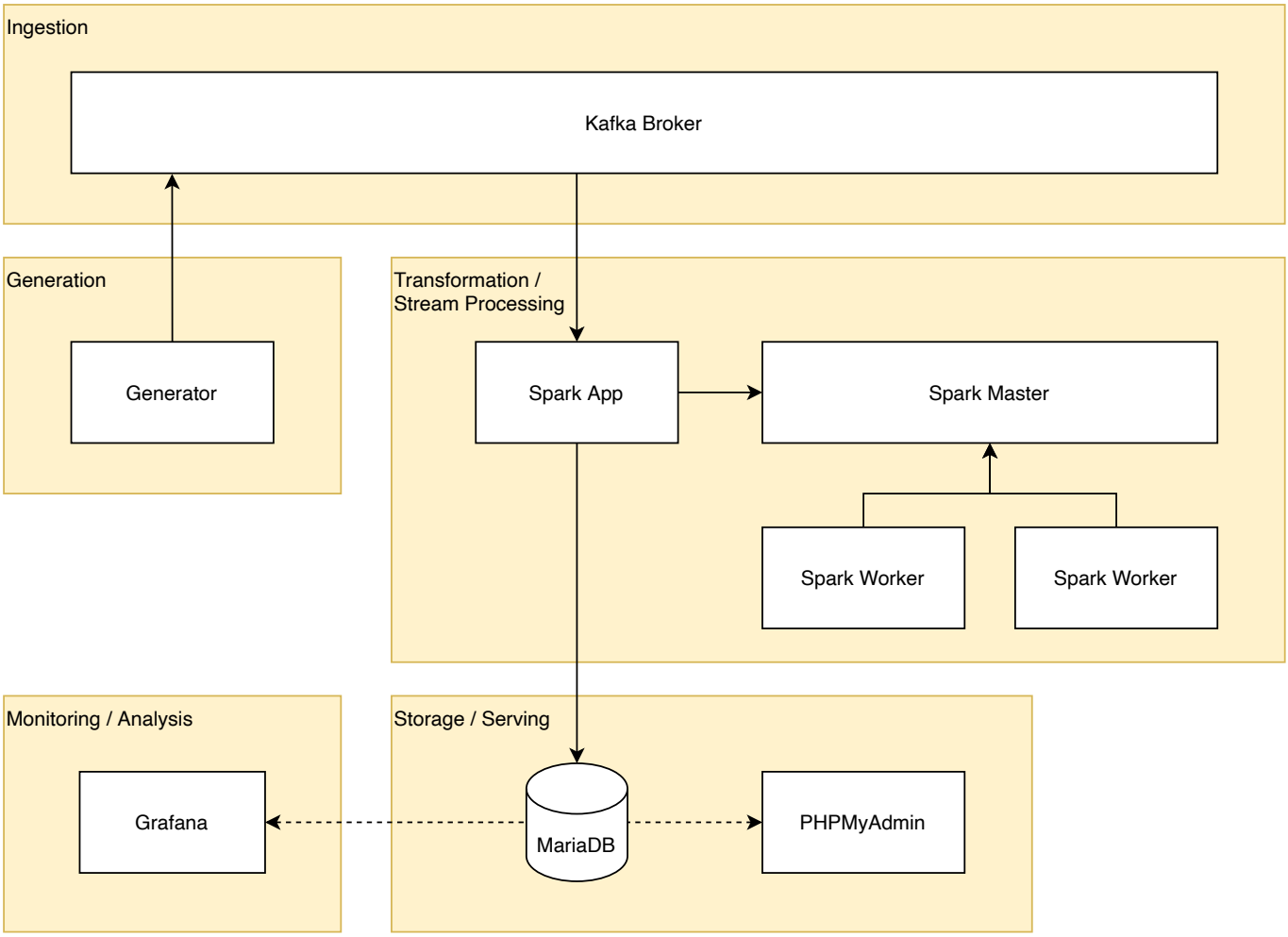
Idea

Climate change is becoming an increasingly critical issue, with a particularly noticeable impact on wind patterns. Rising wind speeds and more frequent storms are among the significant consequences of this global phenomenon. In response to this, there is a growing need for accurate and timely wind data to better understand and predict these changes.

One solution is to develop an application that retrieves (almost) real-time wind data from the German Weather Service (Deutscher Wetterdienst, DWD). This data can then be used to generate models for more precise wind forecasts in the future. Such models would be invaluable for various sectors, including agriculture, energy, transportation, and disaster management, which are all highly sensitive to changes in wind patterns.

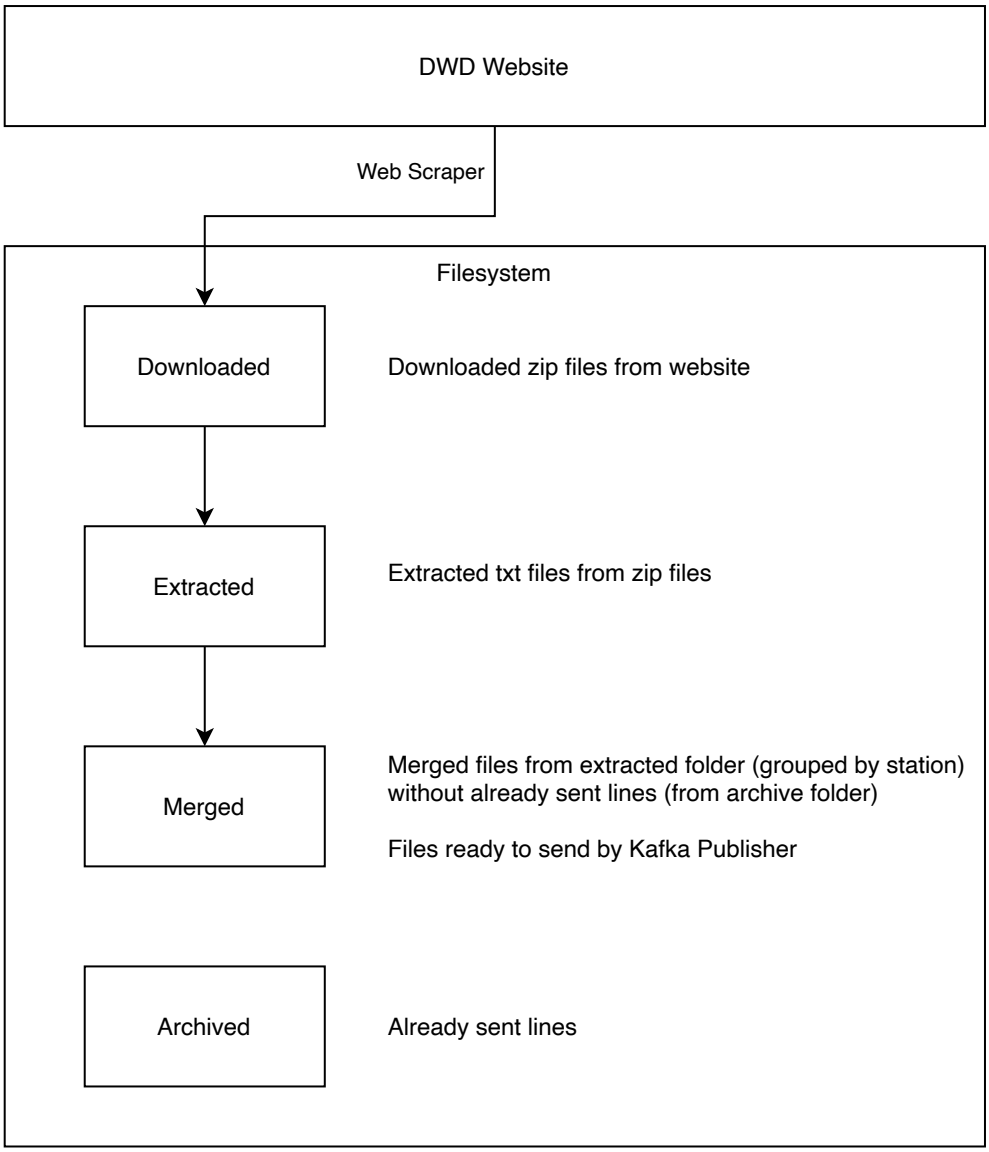
The DWD regularly collects and updates wind data for approximately 300 cities across Germany and makes this information publicly available through its Open Data initiative. By utilizing this open-source data, the application would allow users to access reliable and up-to-date wind information for informed decision-making. Additionally, over time, the data could be analyzed to improve predictive models, contributing to more accurate forecasting and better preparedness for extreme weather events.

Architecture



Python Web Scraper + Streaming Service

Weather and wind data are published by the German weather service (Deutscher Wetterdienst, DWD) in different time periods. The wind data used in this project is published for free as Open Data on this URL. The used dataset contains wind data, which are updated approximately every 30 minutes. The downloadable data contains the current day wind data in 10 minute intervals.



The python application `generator.py` (`generator` in docker) downloads all available files from the above-mentioned website, extracts them and merges relevant csv lines into one single file per weather station. This files can be found in the docker container's temporary folder `/app/tmp/merged`. In order to avoid duplicate lines while sending - especially since the data on the website is only appended to the current-day data, the merged files contain only the lines which were not already sent by the generator. Therefore, the sent lines are stored in the folder `/app/tmp/archived` and are checked during the creation of the merge files.

The downloaded and sent data contain lines in a csv format, always containing the weather station ID, the measurement date, the quality niveau, as well as `F` for the wind force and `D` for the wind direction. `eor` means `end of row`:

```
STATIONS_ID;MESS_DATUM;QN_3;F;D;eor
2667;2024121803;1;5.0;130;eor
```

The quality niveau is described by the DWD as following (see Description of wind data):

- QN = 1 : only formal inspection;
- QN = 2 : checked according to individual criteria;
- QN = 3 : automatic checking and correction.

Apache Kafka as Message Queue

An Apache Kafka broker is used in the cluster as a central message queue. Two topics are used here: `jetstream` for the weather data and `jetstream-description` for the description file of the weather data (in particular, weather station information can be found here).

For reasons of simplicity, no replication is used for the Kafka Broker. The log files are also deleted after 1 gigabyte so that the disk space of the host machine is not overfilled or configuration adjustments (e.g. maximum hard disk space used) are necessary for docker.

Apache Spark for Data Preparation

Apache Spark is used as the central technology for data processing. A Spark master and two Spark workers are first created in the cluster for this purpose. The Bitnami Spark image is used for this. The `docker-compose.yml` file is currently configured to support up to 9 workers. Additional workers require changes in the `replicas` field and range adjustments in `ports`.
The master information is shown on port `8080`, whereas the worker information are shown on the ports specified in the `docker-compose.yml`. The default port range spans up from `8081` to `8089`.

The Bitnami Spark image is also used as the base image for the Spark application, although adjustments still need to be made to make the combination of Spark, Kafka and MariaDB executable.
For this purpose, dependencies are downloaded as JAR files and stored in the `/opt/bitnami/spark/jars/` folder of the `spark-app` container. After installing the Python dependencies, the application file can then be transferred via `spark-submit` and passed to the Spark workers for processing.

In the Spark application, both Kafka topics described above are read as a stream and processed as following:

1. First, the streams are converted into the correct format (conversion of the JSON byte array into CSV, then from CSV into usable columns).
   Incorrect rows, e.g. null values in the station ID or wind speeds <= 0, are removed.
2. Then, a Bucketizer is applied to the data in order to fit the data into bins according to their values.
   The wind direction is binned into buckets for each 45°:

```
0: [0;45[
1: [45;90[
2: [90;135[
3: [135;180[
4: [180;225[
5: [225;270[
6: [270;315[
7: [315;360[
```

Similarly, the wind speed is binned - but into different buckets: low, medium, heavy and extreme wind:

```
0: [0;5[
1: [5;10[
2: [10;17[
3: [17;inf[
```

The bins could be used afterwards for grouping data, performing aggregations, analyses or machine learning operations.

3. Finally, the converted weather and station data are stored in the corresponding staging tables in MariaDB for further use.

The converted weather data is also displayed regularly (but not as frequent as the processing of the database write streams) on the console to provide an overview of whether processing is running or not. The write streams are displayed as a simple `print` statements:

```
-------------------------------------------
Batch: 10
-------------------------------------------
+----------+------------------+-------------+----------+--------------+---------------------+-----------------+
|station_id|measurement_date   |quality_level|wind_speed|wind_direction|wind_direction_bucket|wind_speed_bucket|
+----------+------------------+-------------+----------+--------------+---------------------+-----------------+
|596       |2024-12-31 00:00:00|2            |7.9       |240           |5.0                  |1.0              |
|596       |2024-12-31 00:10:00|2            |7.7       |240           |5.0                  |1.0              |
|596       |2024-12-31 00:20:00|2            |8.6       |230           |5.0                  |1.0              |
|596       |2024-12-31 00:30:00|2            |8.7       |240           |5.0                  |1.0              |
|596       |2024-12-31 00:40:00|2            |8.8       |230           |5.0                  |1.0              |
|596       |2024-12-31 00:50:00|2            |8.8       |240           |5.0                  |1.0              |
|596       |2024-12-31 01:00:00|2            |8.3       |240           |5.0                  |1.0              |
|596       |2024-12-31 01:10:00|2            |8.0       |240           |5.0                  |1.0              |
|596       |2024-12-31 01:20:00|2            |8.4       |230           |5.0                  |1.0              |
|596       |2024-12-31 01:30:00|2            |8.3       |230           |5.0                  |1.0              |
|596       |2024-12-31 01:40:00|2            |9.1       |240           |5.0                  |1.0              |
|596       |2024-12-31 01:50:00|2            |8.4       |240           |5.0                  |1.0              |
|596       |2024-12-31 02:00:00|2            |7.7       |240           |5.0                  |1.0              |
|596       |2024-12-31 02:10:00|2            |8.1       |240           |5.0                  |1.0              |
|596       |2024-12-31 02:20:00|2            |8.5       |240           |5.0                  |1.0              |
|596       |2024-12-31 02:30:00|2            |8.3       |240           |5.0                  |1.0              |
|596       |2024-12-31 02:40:00|2            |8.3       |240           |5.0                  |1.0              |
|596       |2024-12-31 02:50:00|2            |8.4       |240           |5.0                  |1.0              |
|596       |2024-12-31 03:00:00|2            |8.6       |240           |5.0                  |1.0              |
|596       |2024-12-31 03:10:00|2            |8.6       |240           |5.0                  |1.0              |
+----------+------------------+-------------+----------+--------------+---------------------+-----------------+
only showing top 20 rows

Batch with 236 elements saved to database table wind_data_staging
24/12/31 14:22:36 WARN ProcessingTimeExecutor: Current batch is falling behind. The trigger interval is 1000 milliseconds, but spent 1532 milliseconds
Batch with 397 elements saved to database table wind_data_staging
24/12/31 14:22:38 WARN ProcessingTimeExecutor: Current batch is falling behind. The trigger interval is 1000 milliseconds, but spent 1327 milliseconds
Batch with 359 elements saved to database table wind_data_staging
24/12/31 14:22:39 WARN ProcessingTimeExecutor: Current batch is falling behind. The trigger interval is 1000 milliseconds, but spent 1257 milliseconds
```

MariaDB as central storage

MariaDB is used for central storage. When MariaDB is initially started, all necessary databases, tables, procedures and events are created via an init file if they do not already exist.

- The `stations` table contains all the information on the weather stations from the description file:

```
station_id bigint not null,
date_from timestamp not null,
date_until timestamp not null,
height smallint,
latitude double,
longitude double,
name text,
state text,
delivery text
```

Example entry:

station_id	date_from	date_until	height	latitude	longitude	name	state	delivery
15547	2016-10-01 00:00:00	2024-12-30 00:00:00	214	49.1259	9.1428	Heilbronn/Neckar	Baden-Württemberg	Frei

- The `wind_data` table contains all converted and filtered data. This is where the history is built up:

```
station_id bigint not null,
measurement_date timestamp not null,
quality_level tinyint,
wind_speed double comment 'wind speed in m/s',
wind_direction smallint comment 'wind direction in degree',
wind_direction_bucket double comment [...],
wind_speed_bucket double comment [...]
```

Example entry:

station_id	measurement_date ▲ 1	quality_level QN = 1 : only formal inspection; …	wind_speed wind speed in m/s	wind_direction wind direction in degree	wind_direction_bucket Buckets: 0: [0;45[1: [45;90[…	wind_speed_bucket Buckets: 0: [0;5[1: [5;10[…
15547	2024-12-31 00:00:00	3	2.2	220	4	0
15547	2024-12-31 00:10:00	3	2	220	4	0
15547	2024-12-31 00:20:00	3	1	220	4	0
15547	2024-12-31 00:30:00	3	1.2	220	4	0
15547	2024-12-31 00:40:00	3	0.9	190	4	0
15547	2024-12-31 00:50:00	3	1.1	170	3	0
15547	2024-12-31 01:00:00	3	0.4	190	4	0
15547	2024-12-31 01:10:00	3	0.4	190	4	0
15547	2024-12-31 01:20:00	3	1.6	200	4	0
15547	2024-12-31 01:30:00	3	1.7	200	4	0
15547	2024-12-31 01:40:00	3	1	210	4	0
15547	2024-12-31 01:50:00	3	1.1	150	3	0
15547	2024-12-31 02:00:00	3	1.2	210	4	0
15547	2024-12-31 02:10:00	3	1.1	160	3	0
15547	2024-12-31 02:20:00	3	0.8	190	4	0
15547	2024-12-31 02:30:00	3	1.4	200	4	0
15547	2024-12-31 02:40:00	3	1	210	4	0
15547	2024-12-31 02:50:00	3	0.8	170	3	0
15547	2024-12-31 03:00:00	3	1.3	240	5	0
15547	2024-12-31 03:10:00	3	1.6	220	4	0

All tables have corresponding staging tables, since the data can be corrected after some time by the DWD. Those corrections would be inserted in the next iteration but fail on the insert because of the primary key constraints on the database.
Therefore, a staging area is implemented containing no constraints. A procedure is run on the database regularly (every 5 seconds) transferring the staged data to the production tables, performing an upsert operation for colliding rows:

```
begin
    -- Transfer and upsert stations
    insert into stations (station_id, date_from, date_until, height,
latitude, longitude, name, state, delivery)
    select s.station_id, s.date_from, s.date_until, s.height, s.latitude,
s.longitude, s.name, s.state, s.delivery
    from stations_staging s
    on duplicate key update
        date_from = values(date_from),
        date_until = values(date_until),
        height = values(height),
        latitude = values(latitude),
        longitude = values(longitude),
        name = values(name),
        state = values(state),
        delivery = values(delivery);

    -- Delete transferred rows from staging table
    delete from stations_staging ss
```

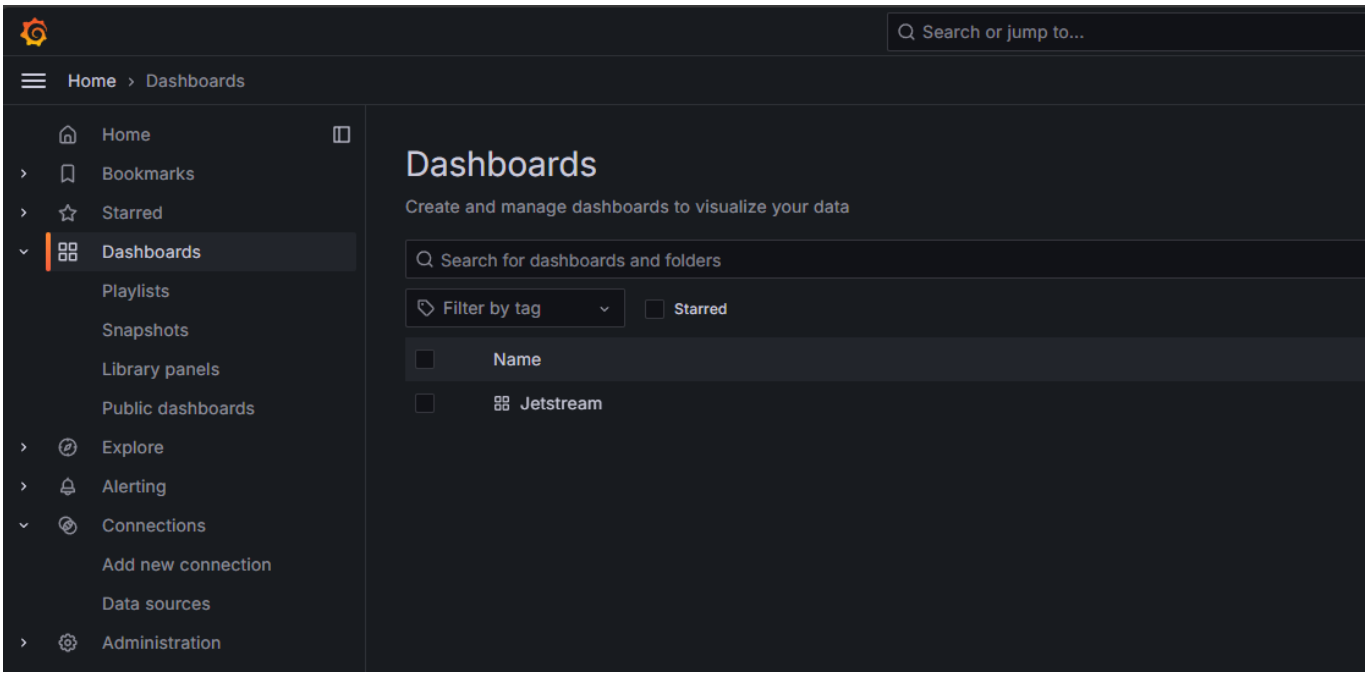
```
    where exists (
        select *
        from stations s
        where s.station_id = ss.station_id
          and s.date_from = ss.date_from
          and s.date_until = ss.date_until
          and s.height = ss.height
          and s.latitude = ss.latitude
          and s.longitude = ss.longitude
          and s.name = ss.name
          and s.state = ss.state
          and s.delivery = ss.delivery
    );

    -- Transfer and upsert wind_data
    insert into wind_data (station_id, measurement_date, quality_level,
wind_speed, wind_direction, wind_direction_bucket, wind_speed_bucket)
    select ws.station_id, ws.measurement_date, ws.quality_level,
ws.wind_speed, ws.wind_direction, ws.wind_direction_bucket,
ws.wind_speed_bucket
    from wind_data_staging ws
    on duplicate key update
        quality_level = values(quality_level),
        wind_speed = values(wind_speed),
        wind_direction = values(wind_direction),
        wind_direction_bucket = values(wind_direction_bucket),
        wind_speed_bucket = values(wind_speed_bucket);

    -- Delete transferred rows from staging table
    delete from wind_data_staging ws
    where exists (
        select *
        from wind_data w
        where w.station_id = ws.station_id
          and w.measurement_date = ws.measurement_date
          and w.quality_level = ws.quality_level
          and w.wind_speed = ws.wind_speed
          and w.wind_direction = ws.wind_direction
          and w.wind_direction_bucket = ws.wind_direction_bucket
          and w.wind_speed_bucket = ws.wind_speed_bucket
    );
end;
```

PHPMyAdmin as web view for MariaDB

In order to be able to view the stored data, PHPMyAdmin is available on port `8090`.

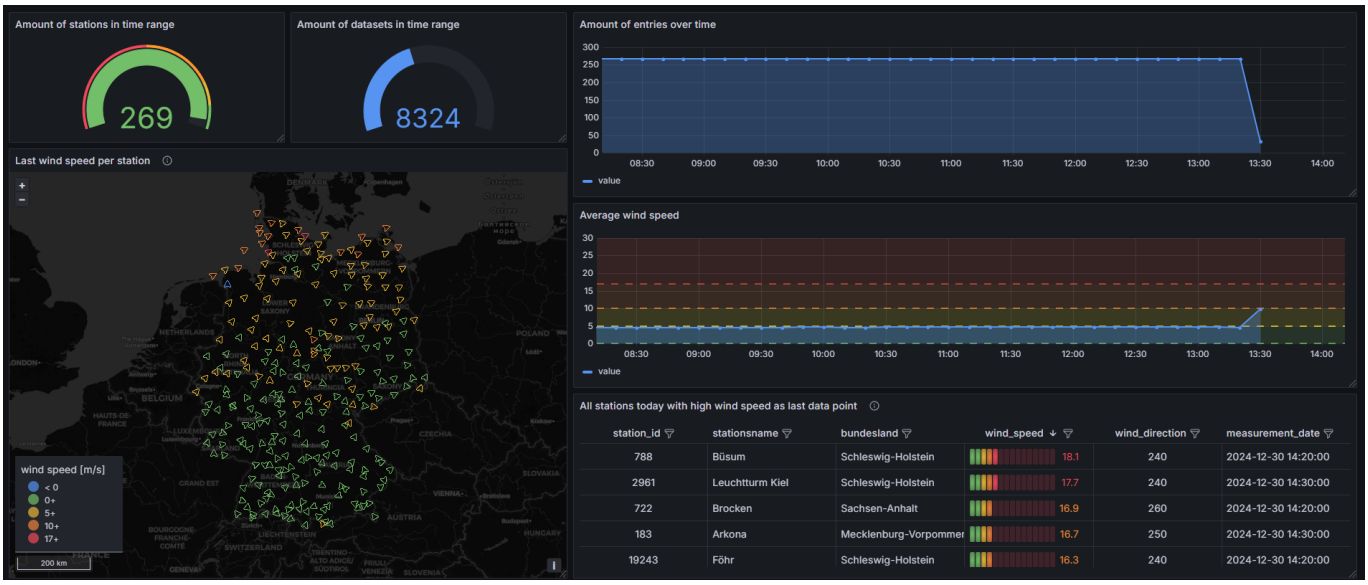
Grafana for monitoring

Grafana is used to create dashboards. A custom dashboard is selected by default, which can display various data. Grafana can be accessed on port `3000`. The default username and password is `jetstream`. The default dashboard `Jetstream` can now be selected under Dashboards on the left-hand side.



The following data are displayed here - reacting responsively to the time range selection on the top:

Feature	Description
Amount of stations in time range	Shows the amount of all stations saved in the database
Amount of datasets in time range	Shows the amount of wind data entries for all stations
Amount of entries over time	Shows a time series of the amount of wind data entries for all stations
Last wind speed per station	Displays a GeoMap containing all stations. The stations are color-coded according to wind speed and oriented according to the wind direction
Average wind speed	Shows a time series of the average wind speed for all stations
All stations today with high wind speed as last data point	Shows the last wind speed for all stations which have a wind speed over 10 as their last data point (but only if the last data point is today)



Of course, the data are changed live as soon as new data are available in the database. This happens approximately every 30 minutes because of the update cycle of the DWD.

Development

- Build + create container: `docker-compose build --no-cache; docker-compose up -d`
- Remove everything: `docker-compose down -v`
- Restart: `docker-compose down; docker-compose build; docker-compose up -d`
- Show logs: `docker logs {container}`

MariaDB:

- Create connection to MariaDB server:
  - Terminal: `docker exec -ti jetstream-mariadb-1 mariadb -u root -p jetstream`
  - Attached shell to mariadb container: `mariadb -u root -p jetstream`
- Show all databases: `show databases;`
- Select database: `use jetstream;`
- Show all tables: `show tables;`
- Queries: `select * from wind_data;`

Screencast

In the screencast all functionalities and features are shown: Link to German screencast