Koninklijk Nederlands
Meteorologisch Instituut
Ministerie van Infrastructuur en Waterstaat

RODEO WP5 EDR API Workshop

Zagreb

10-2025

Jeffrey Vervoort

Welcome!

- Slides:
  - https://tinyurl.com/edr-workshop
- GitHub Repo
  - https://github.com/EUMETNET/ogc-edr-workshop

- Feedback welcome!

Schedule and outline – Day 1

Time	Item
09:00	Welcome
09:10	Introduction of participants
09:20	Introduction to RODEO
09:40	Develop your own EDR
10:50	Coffee break
11:05	Develop your own EDR
12:45	Lunch break
13:35	Develop your own EDR
15:15	Coffee break
15:30	Develop your own EDR
17:00	End of day

Step	Item
0	Setup environment
1	Landing page
2	Retrieve data for single location
3	Filtering (time and parameters)

Schedule and outline – Day 2

Time	Item
09:00	Welcome
09:05	Develop your own EDR
10:45	Coffee break
11:00	Develop your own EDR
12:40	Lunch break
13:30	Develop your own EDR
15:10	Coffee break
15:25	Develop your own EDR
17:00	End of day

Step	Item
3	Filtering (time and parameters)
4	Collection metadata
5	(Bonus) List of locations
6	(Bonus) Area query
	Connecting your data

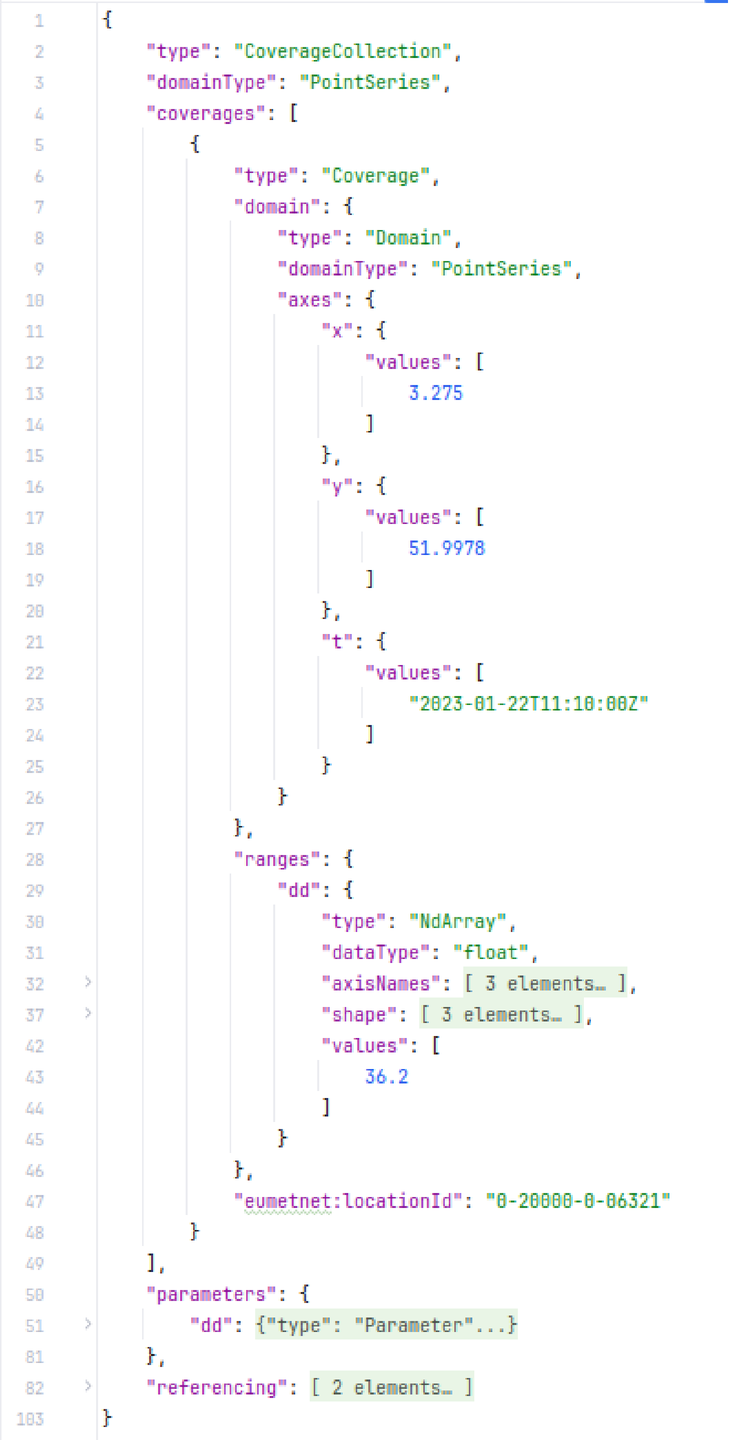
Why OGC EDR?

- Environmental Data Retrieval
- OGC standard
  - Open Geospatial Consortium
- Since 2016
- Discoverable
- Filtering in space and time
- Multiple datasets
  - Collections and instances
- Used by RODEO E-SOH, WP5, ...

Arnoud Apituley

CoverageJSON

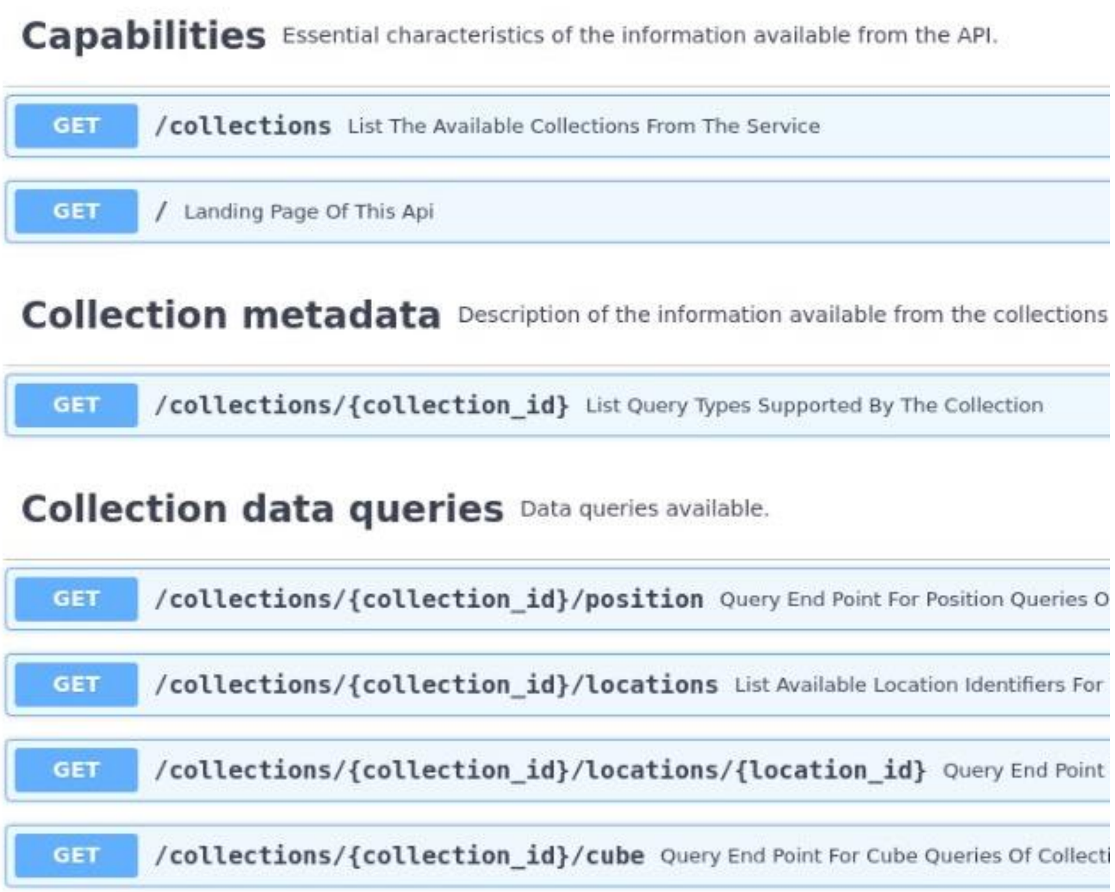
- Recommended EDR output
- A format for publishing geotemporal data (on the web)
  - Time series
  - Gridded
- Format: JSON
  - Easier than NetCDF?
  - Better than CSV
- More info:
  - https://covjson.org/



```
{
    "type": "CoverageCollection",
    "domainType": "PointSeries",
    "coverages": [
        {
            "type": "Coverage",
            "domain": {
                "type": "Domain",
                "domainType": "PointSeries",
                "axes": {
                    "x": {
                        "values": [
                            3.275
                        ]
                    },
                    "y": {
                        "values": [
                            51.9978
                        ]
                    },
                    "t": {
                        "values": [
                            "2023-01-22T11:10:00Z"
                        ]
                    }
                }
            },
            "ranges": {
                "dd": {
                    "type": "NdArray",
                    "dataType": "float",
                    "axisNames": [ 3 elements… ],
                    "shape": [ 3 elements… ],
                    "values": [
                        36.2
                    ]
                }
            },
            "eumetnet:locationId": "0-20000-0-06321"
        }
    ],
    "parameters": {
        "dd": {"type": "Parameter"...}
    },
    "referencing": [ 2 elements… ]
}
```

How to use an EDR API?

- Any HTTPS client
  - e.g. from web browsers to Python
  - Insomnia or Postman



Capabilities Essential characteristics of the information available from the API.

GET **/collections** List The Available Collections From The Service

GET **/** Landing Page Of This Api

Collection metadata Description of the information available from the collections

GET **/collections/{collection_id}** List Query Types Supported By The Collection

Collection data queries Data queries available.

GET **/collections/{collection_id}/position** Query End Point For Position Queries O

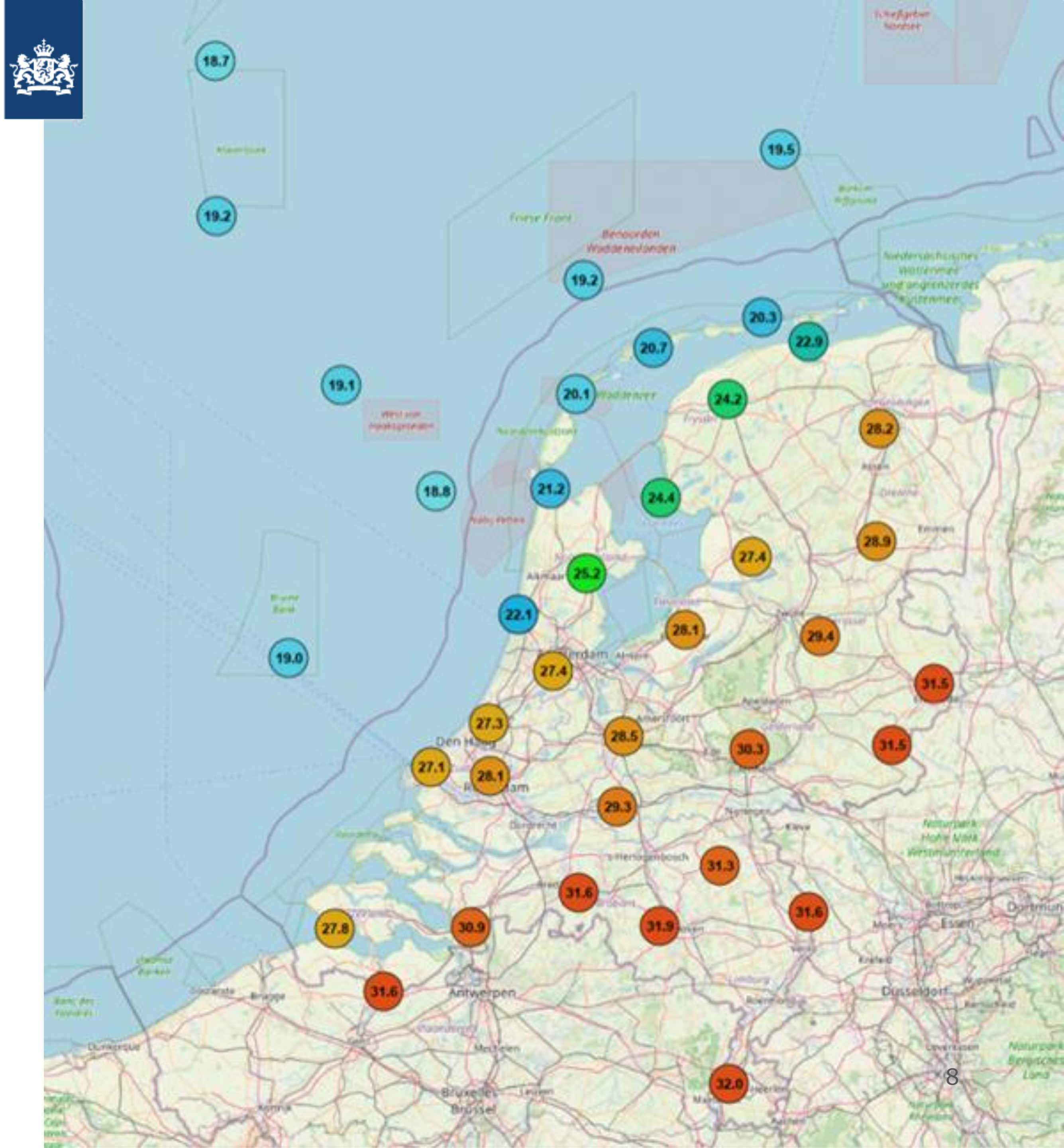
GET **/collections/{collection_id}/locations** List Available Location Identifiers For

GET **/collections/{collection_id}/locations/{location_id}** Query End Point

GET **/collections/{collection_id}/cube** Query End Point For Cube Queries Of Collecti

Demo of KNMI EDR API Synoptic Observations

- https://tinyurl.com/edr-workshop
- Getting started yourself
  1. Go to the KNMI Developer Portal to create an account.
  2. Request an API key for the EDR API.
  3. Go to the EDR documentation KNMI EDR Documentation.
  4. Retrieve the temperature between 12:00 and 14:00 on the 2nd of July for Maastricht for the hourly dataset.
  5. View the result on the CovJSON playground



Step 0

Setting up

Code and environment

- Clone the repository and checkout step_0

```
git clone https://github.com/EURODEO/ogc-edr-workshop.git
git checkout step_0
```

- Python3 virtual environment

```
python3 -m venv venv/
source venv/bin/activate
```

- Install dependencies

```
pip3 install pip-tools
pip-sync
```

Check setup

```
python3 data/data.py
```

- NetCDF datastore
  - daily observation data for the year 2024.
- data.py is interface between data and EDR
- Should be replaced with your data backend (after workshop...)

```
python3 main.py
```

- Starts FastAPI using uvicorn as gateway
- Can be used for step debugging

```
uvicorn main:app --reload
```

- Starts uvicorn with auto reloading
- Test Swagger: http://localhost:8000/docs

How to build an EDR API?

- How ever you want!
  - OGC EDR specification
- Python packages for building EDR APIs
  - Pydantic models for CoverageJSON
  - Pydantic models for EDR
- WP5 Climate EDR:
  - OMSZ EDR API (Hugarian Meteo Service)
  - EDR ANM (Romanian Meteo Service)
- Other examples:
  - EDR-isobaric (MetNorway)
  - RODEO E-SOH EDR API

Jeroen van Schouten

Step 1

Landing Page

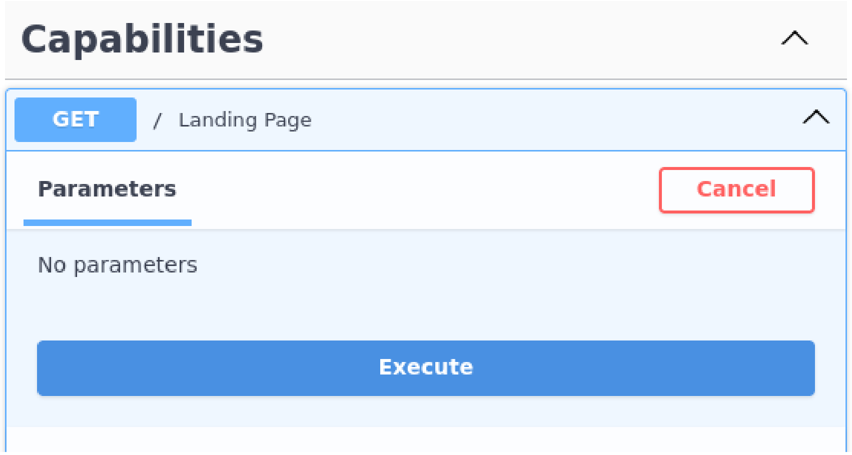
Landing page

- Specification
- Use LandingPageModel from edr_pydantic in main.py

```
from edr_pydantic.capabilities import LandingPageModel

@app.get(
    "/",
    tags=["Capabilities"],
    name="Landing page of this API",
    description="The landing page provides links to the API definition,"
    + " the Conformance statements and the metadata about the feature data in
this dataset.",
    response_model=LandingPageModel,
    response_model_exclude_none=True,
)
async def landing_page(request: Request) -> LandingPageModel:
    pass
```

- Test result http://localhost:8000/
- Or via Swagger



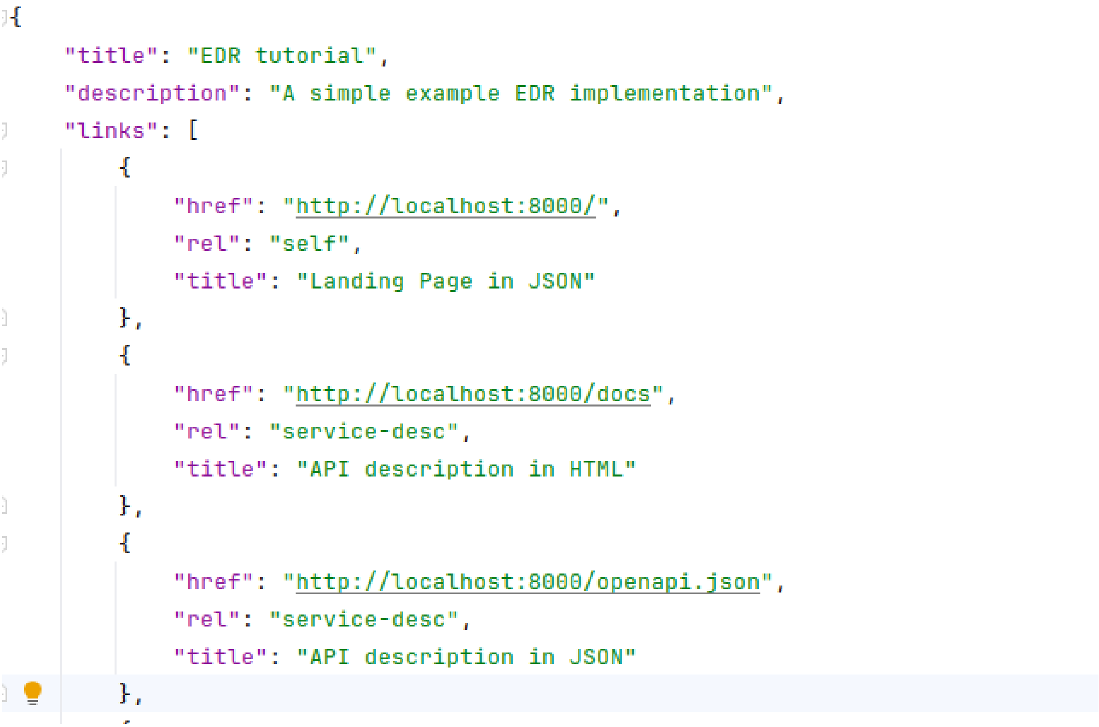
Capabilities

GET / Landing Page

Parameters

No parameters

Result



```
{
    "title": "EDR tutorial",
    "description": "A simple example EDR implementation",
    "links": [
        {
            "href": "http://localhost:8000/",
            "rel": "self",
            "title": "Landing Page in JSON"
        },
        {
            "href": "http://localhost:8000/docs",
            "rel": "service-desc",
            "title": "API description in HTML"
        },
        {
            "href": "http://localhost:8000/openapi.json",
            "rel": "service-desc",
            "title": "API description in JSON"
        },
```

- Problems?

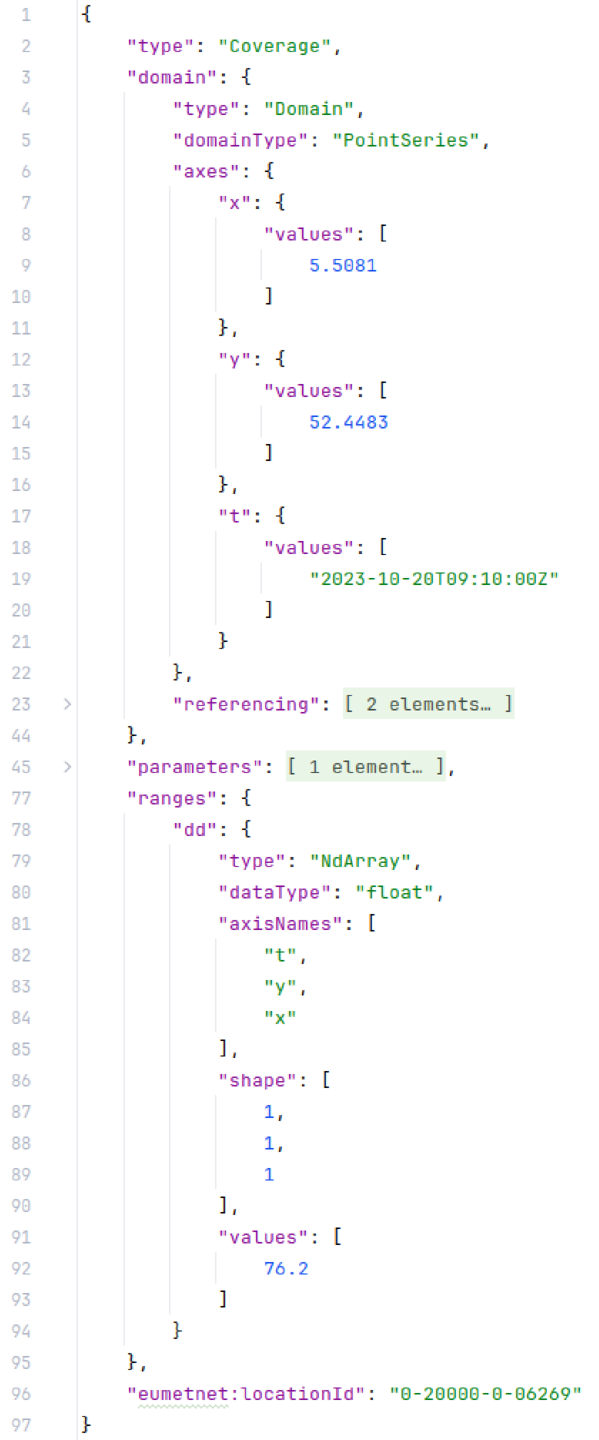
```
git checkout step_1
```

Step 2

Retrieve data for single location

CoverageJSON

- Specification
- Domain (axes)
  - Standard domains
  - We will use PointSeries
- Parameters
- Ranges



```json
{
    "type": "Coverage",
    "domain": {
        "type": "Domain",
        "domainType": "PointSeries",
        "axes": {
            "x": {
                "values": [
                    5.5081
                ]
            },
            "y": {
                "values": [
                    52.4483
                ]
            },
            "t": {
                "values": [
                    "2023-10-20T09:10:00Z"
                ]
            }
        },
        "referencing": [ 2 elements… ]
    },
    "parameters": [ 1 element… ],
    "ranges": {
        "dd": {
            "type": "NdArray",
            "dataType": "float",
            "axisNames": [
                "t",
                "y",
                "x"
            ],
            "shape": [
                1,
                1,
                1
            ],
            "values": [
                76.2
            ]
        }
    },
    "eumetnet:locationId": "0-20000-0-06269"
}
```

Endpoint: /collections/daily/locations/{id}

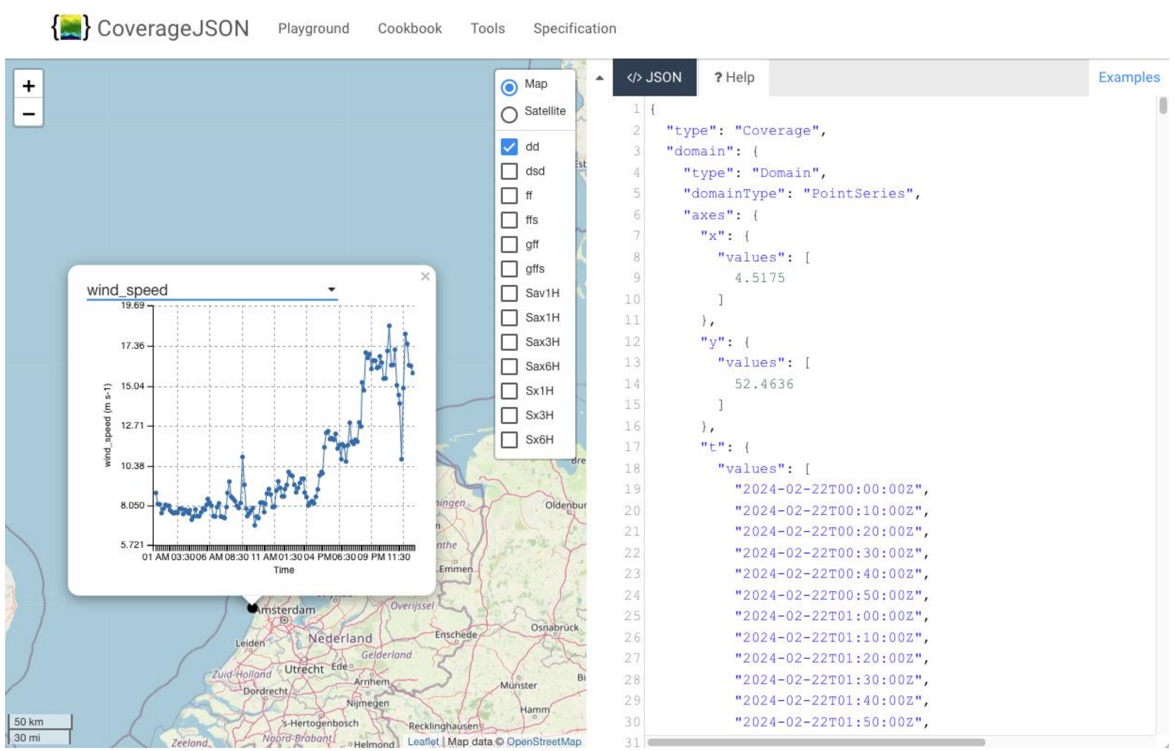
- In `api/observations.py`

```
@router.get(
    "/locations/{location_id}",
    tags=["..."],
    name="...",
    description="...",
    response_model=CoverageCollection,
    response_model_exclude_none=True,
    response_class=CoverageJsonResponse,
)
async def get_data_location_id(
    location_id: Annotated[str, Path(example="0-20000-0-06260")],
    parameter_name: Annotated[
        str | None,
        Query(alias="parameter-name", description="Comma separated list of parameter names.", example="FG, DDVEC"),
    ] = None,
    datetime: Annotated[str | None, Query(example="2024-02-22T00:00:00Z/2024-02-27T00:00:00Z")] = None,
) -> CoverageCollection:
    pass
```

- Use
  - data.get_station()
  - data.get_variables_for_station()
  - data.get_data()
- Remarks:
  - No (or minimal) error handling yet
  - No filtering yet

Result

- Use CovJSON playground



```
{
  "type": "Coverage",
  "domain": {
    "type": "Domain",
    "domainType": "PointSeries",
    "axes": {
      "x": {
        "values": [
          4.5175
        ]
      },
      "y": {
        "values": [
          52.4636
        ]
      },
      "t": {
        "values": [
          "2024-02-22T00:00:00Z",
          "2024-02-22T00:10:00Z",
          "2024-02-22T00:20:00Z",
          "2024-02-22T00:30:00Z",
          "2024-02-22T00:40:00Z",
          "2024-02-22T00:50:00Z",
          "2024-02-22T01:00:00Z",
          "2024-02-22T01:10:00Z",
          "2024-02-22T01:20:00Z",
          "2024-02-22T01:30:00Z",
          "2024-02-22T01:40:00Z",
          "2024-02-22T01:50:00Z",
```

- Problems?

```
git checkout step_2
```

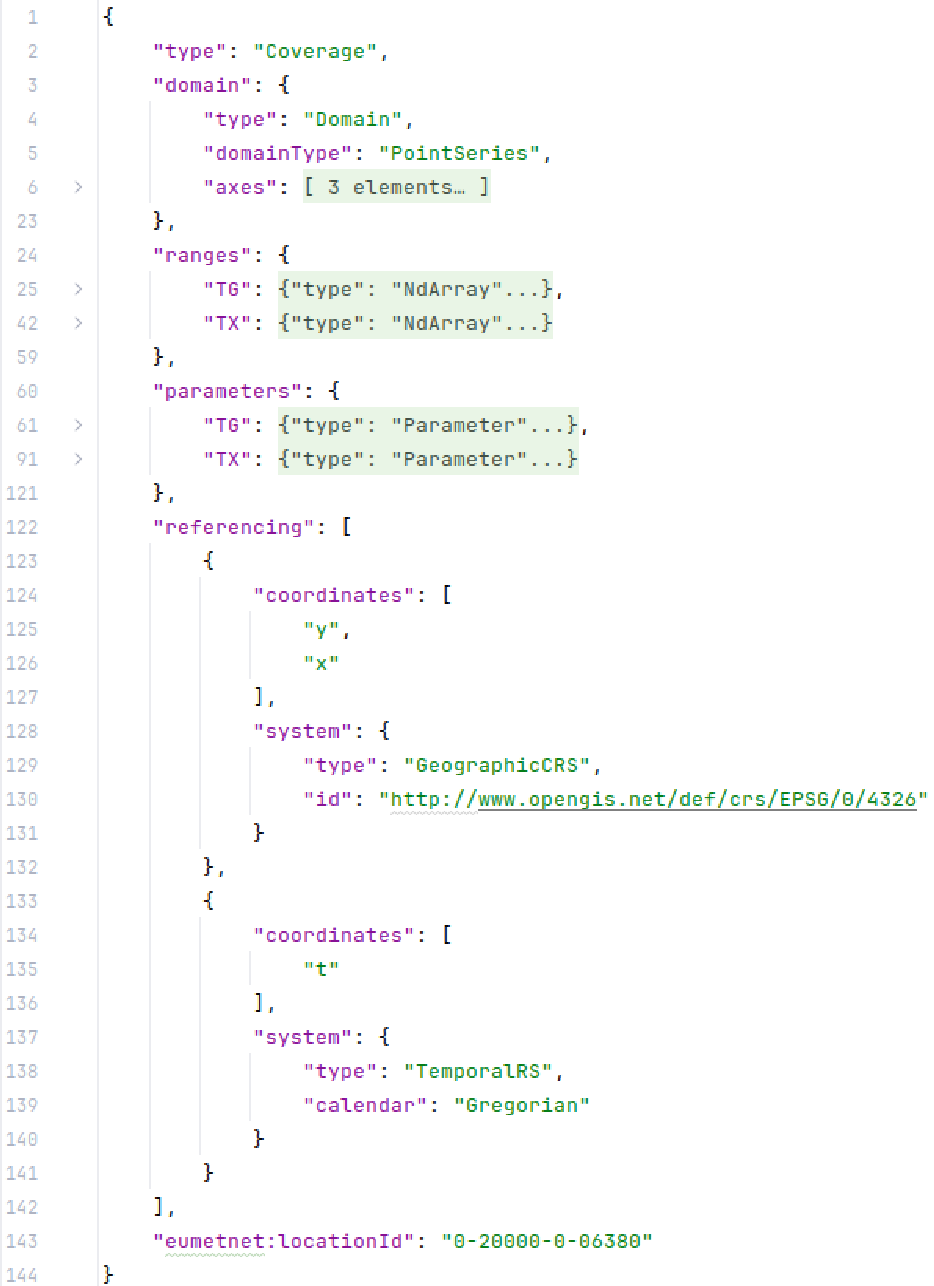
Hints

- Retrieve a single station metadata with get_station(location_id)
- Build the Coverage by retrieving the values with get_data() use them to
  - Create a time axis from the get_data()
  - Create a domain by filling it with the time axis and the coordinates of the station
  - Create the ranges with the values from the get_data() function call
  - Get the locationId from the station and pass it as extra parameter to the Coverage
- Convert the parameters to the CoverageJSON format
  - Get the parameters with the function get_variables_for_station(location_id)
  - Convert the variable to a covjson pydantic parameter by importing the class and filling in the fields
- Either return a Coverage by
  - Adding the parameters
- Or return a CoverageCollection by
  - Passing the Coverage in a list to the CoverageCollection
  - Setting the domain type to point series
  - Set the parameters on CoverageCollection instead of Coverage
  - Set the referencing with the get_reference_system() function call from api.util

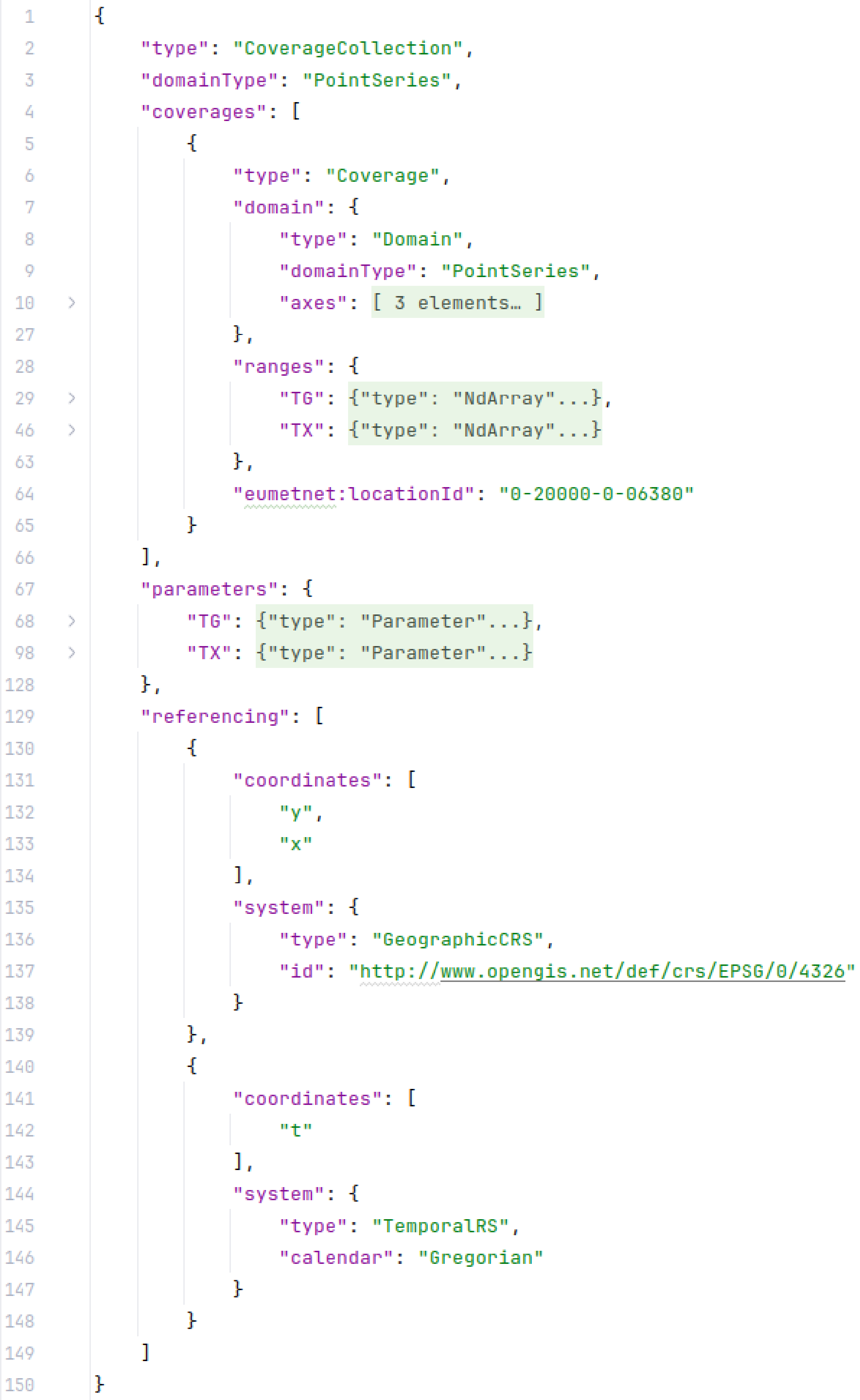
Intermezzo 1

- Coverage vs CoverageCollection
- Metadata in database or as code
- RODEO parameter metadata

Coverage vs CoverageCollection



```
{
    "type": "Coverage",
    "domain": {
        "type": "Domain",
        "domainType": "PointSeries",
        "axes": [ 3 elements… ]
    },
    "ranges": {
        "TG": {"type": "NdArray"...},
        "TX": {"type": "NdArray"...}
    },
    "parameters": {
        "TG": {"type": "Parameter"...},
        "TX": {"type": "Parameter"...}
    },
    "referencing": [
        {
            "coordinates": [
                "y",
                "x"
            ],
            "system": {
                "type": "GeographicCRS",
                "id": "http://www.opengis.net/def/crs/EPSG/0/4326"
            }
        },
        {
            "coordinates": [
                "t"
            ],
            "system": {
                "type": "TemporalRS",
                "calendar": "Gregorian"
            }
        }
    ],
    "eumetnet:locationId": "0-20000-0-06380"
}
```



```
{
    "type": "CoverageCollection",
    "domainType": "PointSeries",
    "coverages": [
        {
            "type": "Coverage",
            "domain": {
                "type": "Domain",
                "domainType": "PointSeries",
                "axes": [ 3 elements… ]
            },
            "ranges": {
                "TG": {"type": "NdArray"...},
                "TX": {"type": "NdArray"...}
            },
            "eumetnet:locationId": "0-20000-0-06380"
        }
    ],
    "parameters": {
        "TG": {"type": "Parameter"...},
        "TX": {"type": "Parameter"...}
    },
    "referencing": [
        {
            "coordinates": [
                "y",
                "x"
            ],
            "system": {
                "type": "GeographicCRS",
                "id": "http://www.opengis.net/def/crs/EPSG/0/4326"
            }
        },
        {
            "coordinates": [
                "t"
            ],
            "system": {
                "type": "TemporalRS",
                "calendar": "Gregorian"
            }
        }
    ]
}
```

Metadata in Database or as Code

- Database
  - Already available?
  - Joins/relations
  - Consistency

- Traits of this metadata
  - Mostly static
  - Release cycles
  - Often requested (also in data queries)

- As code
  - Performance
  - Simple version control
  - Easier updates (no DB migrations)

Metadata in Database or as Code

- Database
  - Already available?
  - Joins/relations
  - Consistency

- Traits of this metadata
  - Mostly static
  - Release cycles
  - Often requested (also in data queries)

- As code
  - Performance
  - Simple version control
  - Easier updates (no DB migrations)

- We did not have an existing solution
- Best fit for the use case at KNMI

Parameter metadata (in EDR)

```
"DDVEC": {
    "type": "Parameter",
    "label": "Mean wind direction",
    "description": "Daily vector mean wind direction, representative for 10 meters, in
degrees. Wind direction has been weighted with wind speed during aggregation. 360=north;
90=east; 180=south; 270=west; 0=calm/variable",
    "data-type": "float",
    "unit": {
        "label": "degree",
        "symbol": {
            "value": "°",
            "type": "https://qudt.org/vocab/unit/DEG"
        }
    },
    "observedProperty": {
        "id": "https://vocab.nerc.ac.uk/standard_name/wind_from_direction/",
        "label": "Wind from direction"
    },
    "measurementType": {
        "method": "mean",
        "duration": "-P1D"
    },
    "eumetnet:standard_name": "wind_from_direction",
    "eumetnet:level": 10.0
},
```

CF conventions

More examples

id	label	description	data-type	unit	standard_name	method	duration	level
DDVEC	Mean wind direction	Daily vector mean wind direction, representative for 10 m, in degrees. Wind direction has been weighted with wind speed during aggregation. 360=north; 90=east; 180=south; 270=west; 0=calm/variable	float	degree	wind_from_direction	Mean	-P1D	10
FG	Mean wind speed	Daily mean windspeed, representative for 10 m, in m/s	float	m/s	wind_speed	Mean	-P1D	10
TG	Mean temperature	Daily mean air temperature, measured at 1.5 m, in degrees Celsius	float	°C	air_temperature	Mean	-P1D	1.5
TN	Minimum temperature	Daily minimum temperature, measured at 1.5 m, in degrees Celsius	float	°C	air_temperature	Min	-P1D	1.5
TX	Maximum temperature	Daily maximum temperature, measured at 1.5 m, in degrees Celsius	float	°C	air_temperature	Max	-P1D	1.5
Q	Global solar radiation	Daily global solar radiation, in J/cm²	float	J/cm²	integral_wrt_time_of_surface_downwelling_shortwave_flux_in_air	Sum	-P1D	
RH	Precipitation amount	Daily precipitation amount, in mm; -1 for <0.05 mm	float	mm	precipitation_amount	Sum	-P1D	
PG	Mean sea level pressure	Daily mean sea level pressure , in hPa	float	hPa	air_pressure_at_mean_sea_level	Mean	-P1D	
UG	Mean relative atmospheric humidity	Daily mean relative atmospheric humidity	float	%	relative_humidity	Mean	-P1D	

More information: EUMETNET - Metocean EDR Profile

QUDT

Relative to ground

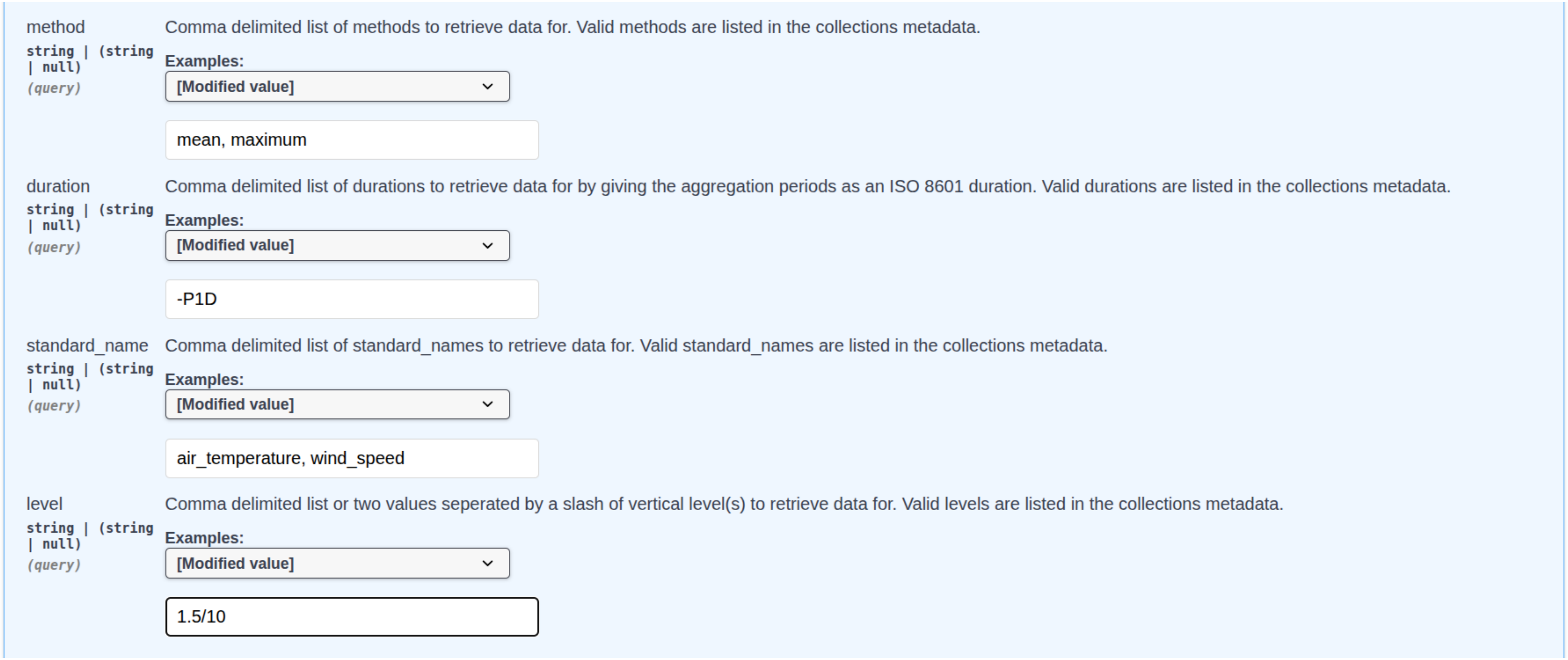
Custom Dimensions

- Retrieve the daily average and maximum air temperature and wind speed measured at heights between 1.5 and 2.0 meters.

- Make use of EDR custom dimensions
  - https://docs.ogc.org/is/19-086r6/19-086r6.html#rc_custom-dimensions-section

- Duration
  - -P1D
- Method
  - mean, maximum
- Standard name
  - air_temperature
- Level
  - 1.5/2.0

Custom Dimensions – Query Parameters



method
`string | (string | null)`
(query)

Comma delimited list of methods to retrieve data for. Valid methods are listed in the collections metadata.

Examples:

[Modified value]

mean, maximum

duration
`string | (string | null)`
(query)

Comma delimited list of durations to retrieve data for by giving the aggregation periods as an ISO 8601 duration. Valid durations are listed in the collections metadata.

Examples:

[Modified value]

-P1D

standard_name
`string | (string | null)`
(query)

Comma delimited list of standard_names to retrieve data for. Valid standard_names are listed in the collections metadata.

Examples:

[Modified value]

air_temperature, wind_speed

level
`string | (string | null)`
(query)

Comma delimited list or two values seperated by a slash of vertical level(s) to retrieve data for. Valid levels are listed in the collections metadata.

Examples:

[Modified value]

1.5/10

Custom Dimensions in metadata

```
"custom": [
  {
    "id": "duration",
    "interval": [
      [
        "-P1D",
        "-P1D"
      ]
    ],
    "values": [
      "-P1D"
    ],
    "reference": "https://en.wikipedia.org/wiki/ISO_8601#Durations"
  },
  {
    "id": "level",
    "interval": [
      [
        0.1,
        10.0
      ]
    ],
    "values": [
      0.1,
      1.5,
      10.0
    ],
    "reference": "Height of measurement above ground level in meters"
  },
  {
    "id": "method",
```

```
    "interval": [
      [
        "maximum",
        "sum"
      ]
    ],
    "values": [
      "maximum",
      "mean",
      "minimum",
      "sum"
    ],
    "reference": "Time aggregation functions"
  },
  {
    "id": "standard_name",
    "interval": [
      [
        "air_pressure_at_mean_sea_level",
        "wind_speed"
      ]
    ],
    "values": [
      "air_pressure_at_mean_sea_level",
      "air_temperature",
      "precipitation_amount",
      "wind_speed",
    ],
    "reference": "https://vocab.nerc.ac.uk/standard_name/"
  }
]
```

Custom Dimensions in metadata

- Collection metadata
- Query parameters
  - E.g. area, cube, {location_id}, position, ...
- Parameter metadata

- Example query:
  - https://api.dataplatform.knmi.nl/edr/v1/collections/daily-in-situ-meteorological-observations-validated/cube?bbox=2.2,49.2,7.2,56.1&datetime=2025-07-02T00:00:00Z/2025-07-04T00:00:00Z
    &duration=-P1D
    &level=1.5/10
    &method=mean,maximum
    &standard_name=air_temperature

Step 3

Filtering (time and parameters)

Filtering (parameters and time)

- Extend `get_data_location_id()`
- Filter `parameter-name`
  - Comma separated list: `FG, DDVEC`
- Filter time
  - start/end (closed interval)
  - ISO8601 string (with Z)

- Remarks:
  - Error handling: What about parameters that don't exist
  - What about parameters that don't exist for the requested station?
- Error responses:
  - 404 for non-existent data (e.g. station does not have data)
  - 400 for mistake in query parameters (e.g. station does not exist because it has a typo)

Result

- Valid parameter values:
  - `200: a filtered response based on the parameters`
- Outside datetime:
  - `404: "detail": "No data available"`
- Non-existent location_id:
  - `400: "detail": "The station {location_id} does not exist."`
- Mistake in parameter-name:
  - `400: "detail": "The following parameters are not available: {'barbecue_weather'}"`

- Problems?

```
git checkout step_3
```

Intermezzo 2

- Input query parameters

Input query parameters

- Multiple approaches are possible in Fastapi/Pydantic
- Aim: reusability
- We tried several
- Work in progress

Jasper van Nieuwenhuizen

Individual parameters

- Simple
- Not reusable
- Manual type conversion

```
@router.get(
    "/locations",
    tags=["Collection data queries"],
    name="List of locations",
    description="List the locations available for the collection",
    response_model=EDRFeatureCollection,
    response_model_exclude_none=True,
    response_class=GeoJsonResponse,
)
async def get_locations(
    bbox: Annotated[str | None, Query(example="5.0,52.0,6.0,52.1")] = None,
    datetime: Annotated[str | None, Query(example="2024-02-22T00:00:00Z/2024-02-27T00:00:00Z")] = None,
    parameter_name: Annotated[
        str | None,
        Query(
            alias="parameter-name",
            description="Comma separated list of parameter names. "
            "Return only locations that have one of these parameter.",
            example="FG, DDVEC",
        ),
    ] = None,
) -> EDRFeatureCollection:
    stations = data.get_stations()

    # Handle bounding box
    if bbox:
        bbox_values = list(map(lambda x: float(str.strip(x)), bbox.split(",")))
        if len(bbox_values) != 4:
            raise HTTPException(status_code=400, detail="If provided, the bbox should have 4 values")
        left, bottom, right, top = bbox_values
        ...
```

Single Pydantic model

- Less duplication
- Doesn’t work!

Issues with:

- Default values
- Examples
- Mix & match

```
class LocationsQueryModel(BaseModel):
    bbox: str | None = Field(None, description="Only features that have a geometry"
                                               " that intersects the bounding box are selected.",
                             examples=["5.0,52.0,6.0,52.1"]),
    datetime: str | None = Field(None, description="Either a date-time or an interval, open or closed.",
                                 examples=["2024-02-22T01:00:00Z/2024-02-22T02:00:00Z"])

@router.get(
    "/locations",
    tags=["Collection data queries"],
    response_model=EDRFeatureCollection,
    response_model_exclude_none=True,
    response_class=GeoJsonResponse,
)
async def get_locations(
       query: Annotated[LocationsQueryModel, Query()]
) -> EDRFeatureCollection:

    # Handle bounding box
    if query.bbox:
        bbox_values = list(map(lambda x: float(str.strip(x)), query.bbox.split(",")))
        if len(bbox_values) != 4:
            raise HTTPException(status_code=400, detail="If provided, the bbox should have 4 values")
        left, bottom, right, top = bbox_values
        ...
```

Custom types

- Automatic type conversion
- Complicated
- Reusable

```python
BBox = Tuple[float, float, float, float]

def _validate_str_to_bbox(value: str) -> BBox:
    if type(value) is str:
        value = tuple(float(x) for x in value.split(","))
    if len(value) != 4:
        raise ValueError("bbox expects 4 values")
    return TypeAdapter(BBox).validate_python(value)


BBoxFromString: TypeAlias = Annotated[BBox, PlainValidator(_validate_str_to_bbox,
                                                           json_schema_input_type=str)]

BBoxQueryOptional = Query(
    description="Only features that have a geometry that intersects the bounding box are selected.",
    openapi_examples={"4 numbers": Example(summary="Bounding box - 2 dimensional", value="5.1, 52.0, 5.2,
52.1")},
)

BBoxOptionalParam = Annotated[BBoxFromString | None, BBoxQueryOptional, WithJsonSchema({"type": "string"})]

@router.get(
    "/locations",
    tags=["Collection data queries"],
    response_model=EDRFeatureCollection,
    response_model_exclude_none=True,
    response_class=GeoJsonResponse,
)
async def get_locations(
        bbox: BBoxOptionalParam = None,
        datetime: DatetimeIntervalOptionalParam = None,
) -> EDRFeatureCollection:
```

Step 4

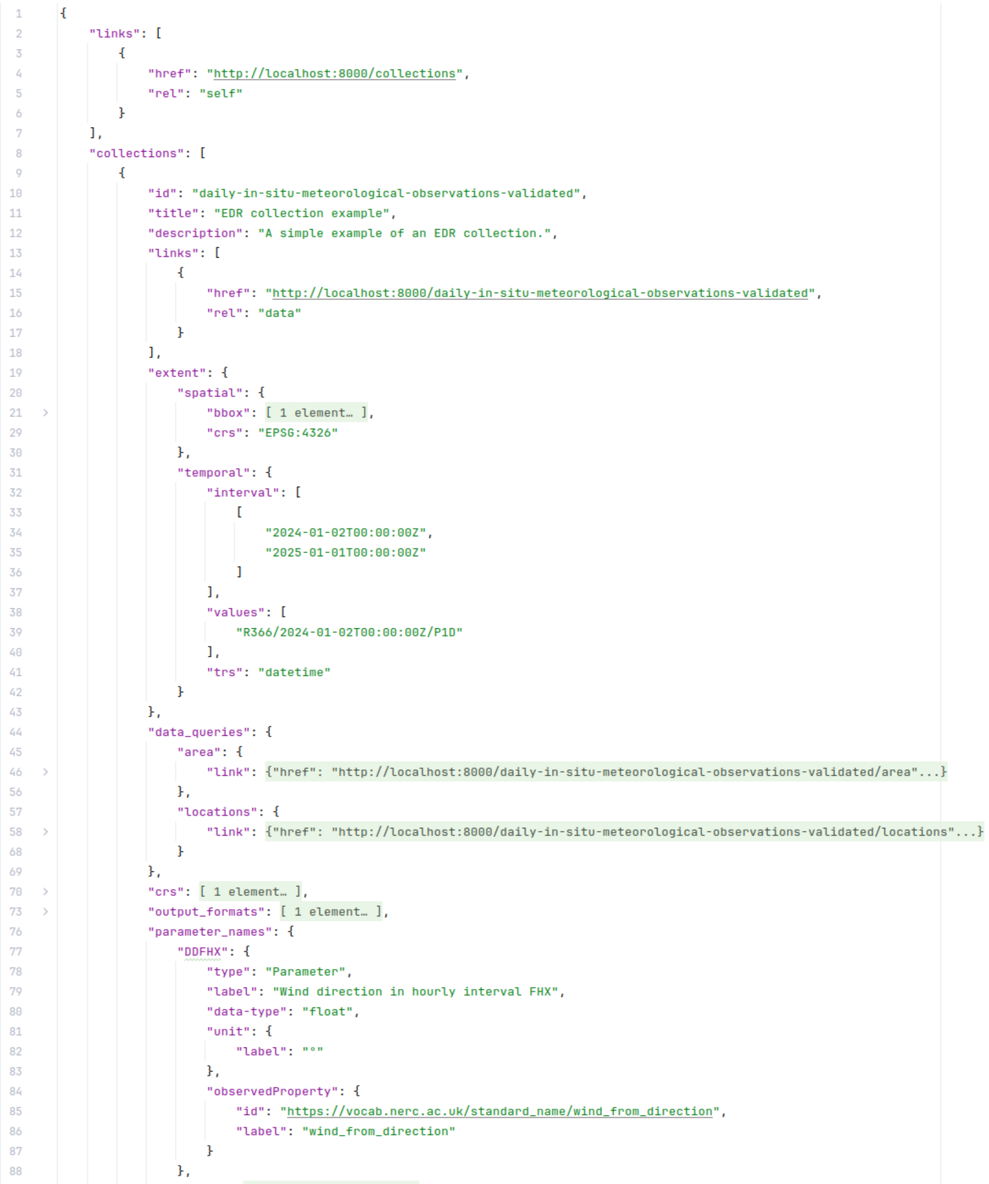
Collection metadata

Collection metadata

- EDR specification
  - Examples in spec
- Parameters EDR vs CovJSON
  - In the EDR Collection they are listed for querying
  - In CovJSON they are detailed and help to interpret the data
- Implement /collections/daily
  - Bonus: Implement /collections

Vasko Lozanov

Result



```json
{
    "links": [
        {
            "href": "http://localhost:8000/collections",
            "rel": "self"
        }
    ],
    "collections": [
        {
            "id": "daily-in-situ-meteorological-observations-validated",
            "title": "EDR collection example",
            "description": "A simple example of an EDR collection.",
            "links": [
                {
                    "href": "http://localhost:8000/daily-in-situ-meteorological-observations-validated",
                    "rel": "data"
                }
            ],
            "extent": {
                "spatial": {
                    "bbox": [ 1 element… ],
                    "crs": "EPSG:4326"
                },
                "temporal": {
                    "interval": [
                        [
                            "2024-01-02T00:00:00Z",
                            "2025-01-01T00:00:00Z"
                        ]
                    ],
                    "values": [
                        "R366/2024-01-02T00:00:00Z/P1D"
                    ],
                    "trs": "datetime"
                }
            },
            "data_queries": {
                "area": {
                    "link": {"href": "http://localhost:8000/daily-in-situ-meteorological-observations-validated/area"...}
                },
                "locations": {
                    "link": {"href": "http://localhost:8000/daily-in-situ-meteorological-observations-validated/locations"...}
                }
            },
            "crs": [ 1 element… ],
            "output_formats": [ 1 element… ],
            "parameter_names": {
                "DDFHX": {
                    "type": "Parameter",
                    "label": "Wind direction in hourly interval FHX",
                    "data-type": "float",
                    "unit": {
                        "label": "°"
                    },
                    "observedProperty": {
                        "id": "https://vocab.nerc.ac.uk/standard_name/wind_from_direction",
                        "label": "wind_from_direction"
                    }
                },
```

- Problems?

```
git checkout step_4
```

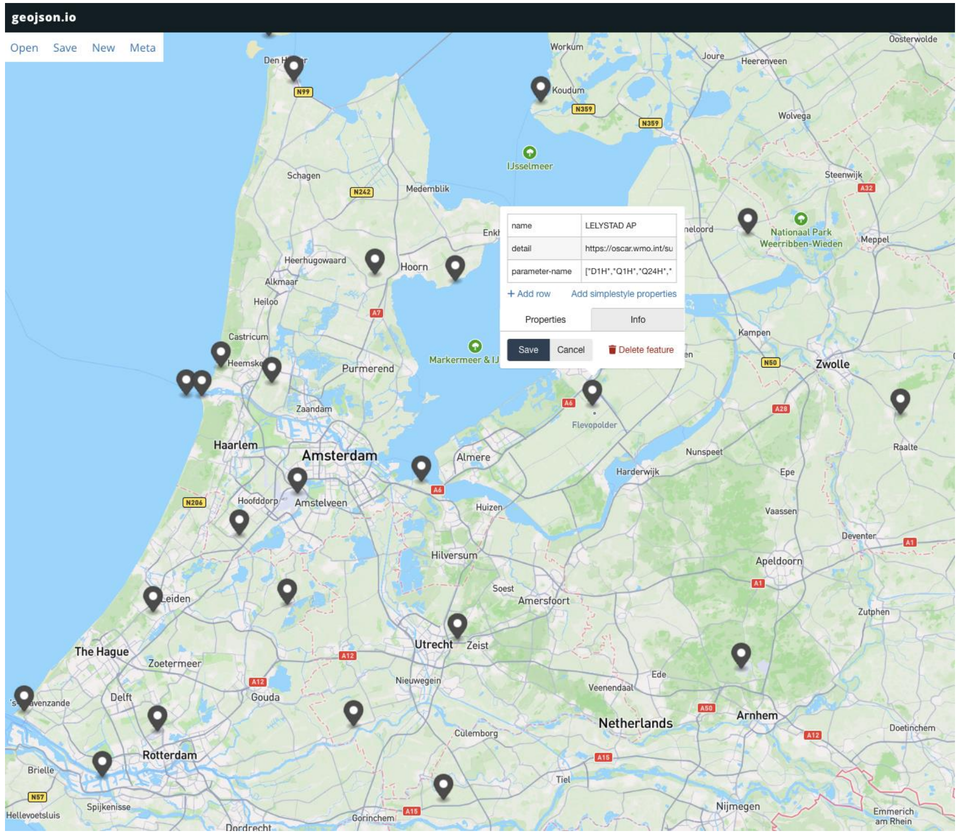
Step 5

List of locations

BONUS

List of locations

- /daily/locations
  - Return GeoJSON
  - geojson.io playground
- Query parameters
  - None in the spec
  - Suggestion: bbox, datetime and parameter-name
- EDRFeatureCollection
  - GeoJSON FeatureCollection with parameters



name	LELYSTAD AP
detail	https://oscar.wmo.int/su
parameter-name	["D1H","Q1H","Q24H","

Step 6

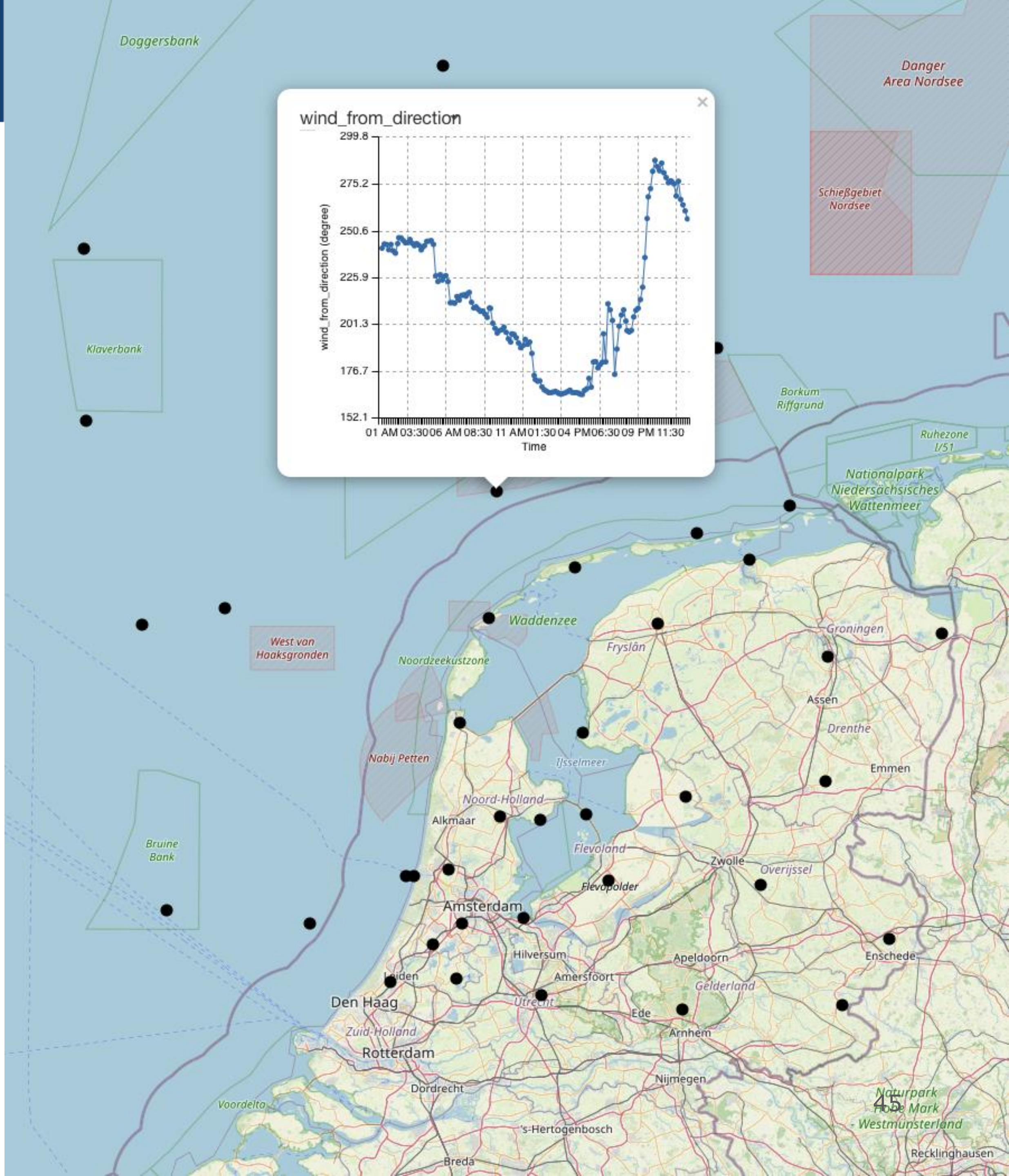
/area query

BONUS

/area query

- CoverageCollection
- Addition of station identifier to Coverage: "eumetnet:locationId": station.wsi
- Hint: calculate stations in polygon: (The code below is not suitable for production as it assumes the world is flat.)

```
from shapely import geometry, wkt
poly = wkt.loads(coords)
stations_in_polygon = [s for s in get_stations()
                              if geometry.Point(s.longitude,
s.latitude).within(poly)]
```

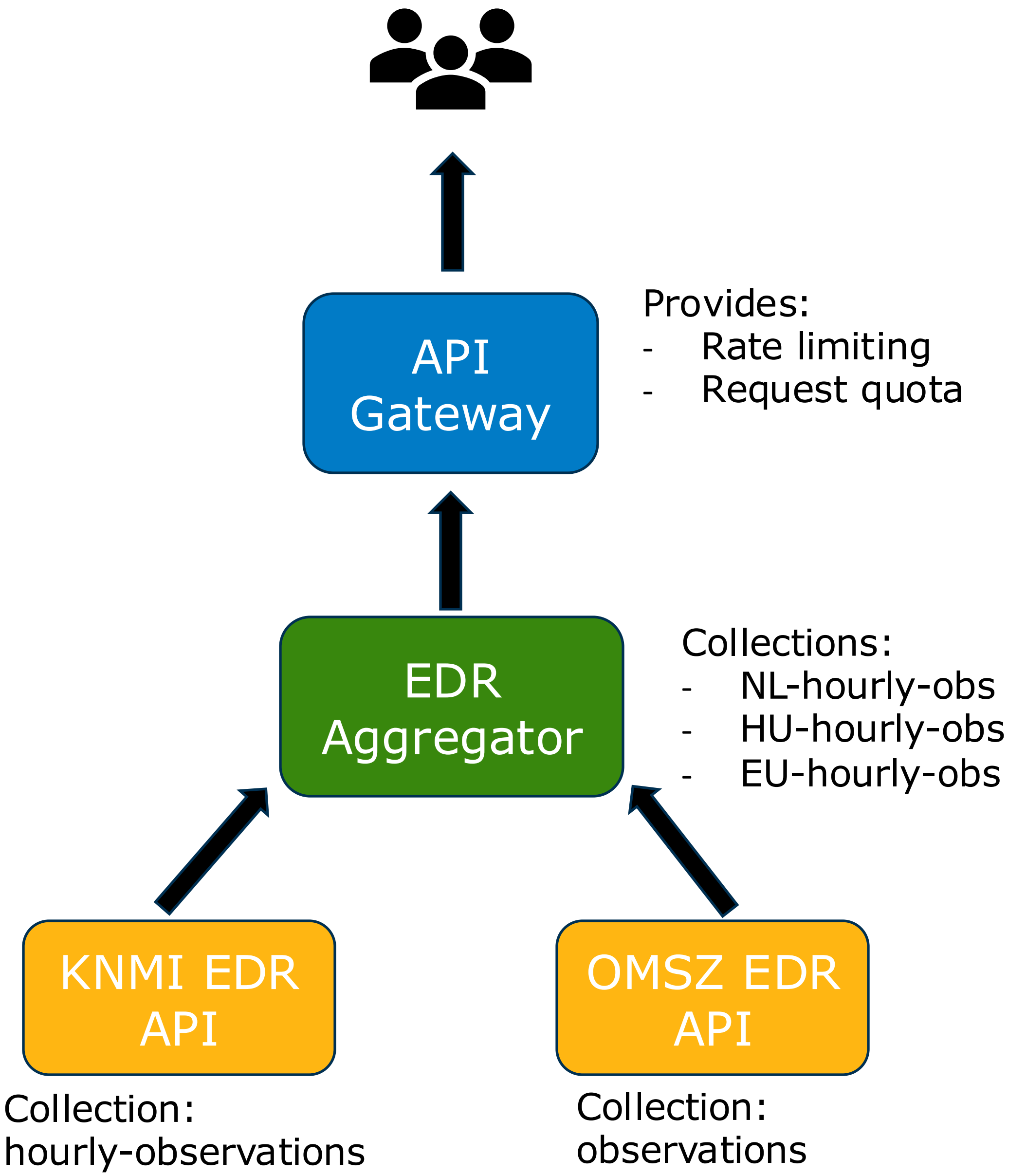


Aggregator

Short intro

Work in Progress: EDR Aggregator

- Unify multiple EDRs into one
- (Proposed) functionality:
  - Output conversion: NetCDF, CSV
  - Caching
  - Queries spanning collections
- Benefits for NMHSs:
  - Simplify EDR implementation and deployment
  - No direct public access required
  - Implementation agnostic
- Benefits for users:
  - One endpoint
  - One query
- Work in progress: To be released as OSS



API Gateway

Provides:
- Rate limiting
- Request quota

EDR Aggregator

Collections:
- NL-hourly-obs
- HU-hourly-obs
- EU-hourly-obs

KNMI EDR API

Collection:
hourly-observations

OMSZ EDR API

Collection:
observations